

## University of Groningen

### Splines for engineers

Barendrecht, Pieter

DOI:  
[10.33612/diss.102688532](https://doi.org/10.33612/diss.102688532)

**IMPORTANT NOTE:** You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

*Document Version*  
Publisher's PDF, also known as Version of record

*Publication date:*  
2019

[Link to publication in University of Groningen/UMCG research database](#)

*Citation for published version (APA):*  
Barendrecht, P. (2019). *Splines for engineers: with selected applications in numerical methods and computer graphics*. [Thesis fully internal (DIV), University of Groningen]. Rijksuniversiteit Groningen. <https://doi.org/10.33612/diss.102688532>

#### Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

#### Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

# **Splines for engineers**

With selected applications in  
numerical methods and computer graphics

With 120 figures

The research for this dissertation was conducted at the Scientific Visualization and Computer Graphics (SVCG) research group, part of the Bernoulli Institute and the Faculty of Science and Engineering, University of Groningen, the Netherlands, as well as at the Basque Center for Applied Mathematics (BCAM), Bilbao, Spain.

Text, illustrations and layout by P.J. Barendrecht (unless otherwise indicated). Typeset in  $\text{\LaTeX}$  using the Linux Libertine font. Printed by ProefschriftMaken.

ISBN 978-94-034-2166-7 (printed version)

ISBN 978-94-034-2165-0 (digital version)



rijksuniversiteit  
 groningen

# Splines for engineers

With selected applications in  
numerical methods and computer graphics

## Proefschrift

ter verkrijging van de graad van doctor aan de  
Rijksuniversiteit Groningen  
op gezag van de  
rector magnificus prof. dr. C. Wijmenga  
en volgens besluit van het College voor Promoties.

De openbare verdediging zal plaatsvinden op

vrijdag 6 december 2019 om 16:15 uur

door

**Pieter Jaap Barendrecht**

geboren op 21 februari 1988  
te Bennekom, gemeente Ede



**Promotor**

Prof. dr. J.B.T.M. Roerdink

**Copromotor**

Dr. J. Kosinka

**Beoordelingscommissie**

Prof. dr. K. Hormann

Prof. dr. G. Vegter

Prof. dr. A.C. Telea

*Dedicated to the development of free and open-source software everywhere.*

## Summary

There are many interesting aspects and applications of splines, in particular in the context of arbitrary manifold topology. Starting from the elementary Bézier curve, the journey towards subdivision surfaces takes us through the steps of smoothly connecting individual segments into B-splines or box splines, the uniform refinement of those splines (i.e. their two-scale relation) and ultimately the tuning of stencil coefficients that allows us to associate meshes of arbitrary manifold topology with surfaces that are globally at least  $C^1$  continuous. The mathematical principles involved in these steps include elegant concepts such as directional convolution, projection of polytopes and (discrete) Fourier transforms.

Regarding subdivision surfaces, we propose two scientific contributions. First, we complete a subdivision scheme based on half-box splines, which provides us with a way to subdivide three-valent meshes of arbitrary manifold topology and define associated piecewise cubic surfaces (composed of triangular patches) that connect with  $C^1$  continuity. It provides us with insight regarding the (artificial) connection of control points into a control net — which in this case generally consists of mostly hexagons — as well as so-called ineffective eigenvectors, which are related to the locally linearly dependent blending functions. It also completes the list of low-degree (box) spline-based subdivision schemes.

Secondly, we study improved quadrature rules for the subdivision splines associated with the Catmull–Clark subdivision scheme. The first approach exploits the  $C^2$  continuity across patch borders to integrate strips composed of multiple patches more efficiently compared to a per-patch integration approach using e.g. Gauss–Legendre quadrature. The integration points and -weights are computed by means of homotopy continuation, though preliminary results show that a purely numerical optimisation approach yields the same results. The latter brings us to a second approach to further optimise the quadrature rules based on non-linear optimisation, which indicates that very efficient rules exist to numerically integrate the subdivision splines to high precision. Both approaches contribute towards overall more efficient numerical simulations when using spline-based methods such as isogeometric analysis on geometries modelled as (Catmull–Clark) subdivision surfaces.

A third contribution, not related to subdivision surfaces, lies in the realm of vector graphics. Here, we show that local refinement of bicubic patches facilitates the creation of resolution-independent illustrations that can be (close to) photorealistic. In addition to this improvement to the gradient mesh primitive, we propose support for more flexible topologies (something which can still be generalised even further) as well as sharp colour transitions. Although web browsers currently lack built-in support to render these types of vector graphics, an implementation in WebGL 2 shows that fast and accurate renderings can indeed be achieved.

In addition to the above, various topics from computer graphics, including (non-uniform) tessellation of quadrilateral and multi-sided domains, as well as (subdivision) shading, are improved upon and applied in these diverse contexts.

## Samenvatting

Splines hebben vele fascinerende kenmerken en toepassingen, vooral in het kader van variëteiten met een arbitraire topologie. De weg van een eenvoudige Bézierkromme naar een onderverdelingsvlak (Engels: *subdivision surface*) leidt ons via glad aaneengeschakelde segmenten, die B-splines of box splines vormen, naar verfijningsrelaties voor deze splines. De verfijningsrelaties kunnen vervolgens worden veralgemeniseerd naar de bovengenoemde variëteiten. Door de bijbehorende coëfficiënten van deze relaties zorgvuldig te kiezen kunnen netwerken of mazen (Engels: *meshes*) met willekeurige connectiviteit worden geassocieerd met oppervlakken die globaal ten minste  $C^1$ -continu zijn. De verschillende stappen in dit proces maken gebruik van diverse wiskundige principes, waaronder richtingsconvolutie, de projectie van polytopen en (discrete) Fouriertransformaties.

Wat betreft onderverdelingsvlakken presenteren we twee wetenschappelijke bijdragen. Allereerst beschouwen we een nieuw schema gebaseerd op half-box splines wat het mogelijk maakt om netwerken waarbij in elk knooppunt drie lijnen samenkomen (en over het algemeen grotendeels uit zeshoeken bestaan) te associëren met stuks-gewijs derdegraads oppervlakken bestaande uit driehoekige segmenten die onderling  $C^1$ -continu verbonden zijn. Het schema vertoont enkele bijzondere eigenschappen, waaronder de zogeheten ineffektieve eigenvectoren die te maken hebben met de plaatselijke lineaire afhankelijkheid van de vormfuncties. Tevens voltooit dit schema de lijst van (op box splines gebaseerde) onderverdelingsschema's van lage graad.

Ten tweede bestuderen we verbeterde kwadratuurformules voor splines die geassocieerd zijn met het Catmull–Clark schema. In de eerste aanpak wordt de  $C^2$ -continuïteit tussen de segmenten gebruikt om groepen van deze segmenten op efficiënte wijze tegelijk te integreren (een verbetering vergeleken met het gebruik van bijvoorbeeld Gauss–Legendre kwadratuur voor elk segment afzonderlijk). Een alternatieve benadering die nog verder moet worden onderzocht is gebaseerd op een puur numerieke aanpak en gebruikt niet-lineaire optimalisatie om integratiepunten en -gewichten te vinden die de splines met hoge nauwkeurigheid numeriek kunnen integreren. De toegenomen efficiëntie is van groot belang voor numerieke methoden zoals isogeometrische analyse voor objecten die als onderverdelingsvlak zijn gemodelleerd.

Ten slotte leveren we een bijdrage in het rijk der vectorgrafiek. We tonen aan dat het plaatselijk verfijnen van elementen in een zogeheten gradiëntenmaas (Engels: *gradient mesh*) het vervaardigen van resolutieonafhankelijke en natuurgetrouwe illustraties sterk vereenvoudigt. Daarnaast stellen we voor om de strenge eisen aan de topologie van een dergelijke maas te verzachten en om de mogelijkheid om de maas plotseling van kleur te veranderen gebruiksvriendelijker te maken. Ondersteuning van webbrowsers voor dergelijke afbeeldingen ontbreekt momenteel nog, al laat een implementatie in WebGL2 zien dat er zeker mogelijkheden zijn.

In aanvulling op de bovenstaande bijdragen worden er enkele onderwerpen uit de computergrafiek besproken en verbeterd, waaronder de (ongelijkmatige) betegeling (Engels: *tessellation*) van vier- en veelhoekige domeinen op de GPU en de belichting van onderverdelingsvlakken.

# Contents

<b>Summary</b>	<b>vi</b>
<b>Samenvatting</b>	<b>vii</b>
<b>Preface</b>	<b>1</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Content and contributions . . . . .	4
1.2 About the writing style . . . . .	5
1.3 About the illustrations . . . . .	6
<b>2 Spline basics</b>	<b>8</b>
2.1 Spline curves . . . . .	10
2.1.1 Bézier curves . . . . .	10
2.1.2 B-spline curves . . . . .	14
2.1.3 Alternative representations . . . . .	19
2.2 Spline patches . . . . .	22
2.2.1 Bézier patches . . . . .	22
2.2.2 B-spline patches . . . . .	25
2.2.3 Simplex splines . . . . .	25
2.2.4 Box splines . . . . .	27
2.2.5 Half-box splines . . . . .	31
2.2.6 Alternative representations . . . . .	33
2.3 Refinement . . . . .	37
2.3.1 Knot insertion . . . . .	37
2.3.2 Two-scale relation . . . . .	40
<b>3 Constructing smooth surfaces of arbitrary manifold topology</b>	<b>44</b>
3.1 $G^1$ composite surfaces . . . . .	46
3.1.1 Interpolating curve networks with Bézier patches . . . . .	46
3.1.2 Interpolating curve networks with Gregory patches . . . . .	58
3.2 Subdivision surfaces . . . . .	61
3.2.1 Half-box spline subdivision . . . . .	63
3.2.2 Catmull–Clark subdivision . . . . .	77
3.2.3 An overview of (half-)box spline-based subdivision schemes . . . . .	81

3.3	Alternatives . . . . .	83
3.3.1	Approximated Catmull–Clark subdivision surfaces . . . . .	83
3.3.2	T-spline surfaces . . . . .	84
4	<b>Spline-based numerical methods</b> . . . . .	<b>86</b>
4.1	The finite element method (FEM) . . . . .	88
4.1.1	Variational formulation . . . . .	88
4.1.2	Discretisation, meshing and element types . . . . .	89
4.1.3	Quadrature and assembly . . . . .	92
4.1.4	Solving the linear system $\mathbf{K}\mathbf{u} = \mathbf{q}$ . . . . .	96
4.1.5	The patch test . . . . .	96
4.1.6	Error estimators and refinement . . . . .	97
4.2	Isogeometric analysis (IgA) . . . . .	100
4.2.1	Subdivision-based IgA . . . . .	103
4.2.2	Improving quadrature for Catmull–Clark elements (I) . . . . .	109
4.2.3	Patchification of the limit surface . . . . .	116
4.2.4	Improving quadrature for Catmull–Clark elements (II) . . . . .	118
4.2.5	Refinement for Catmull–Clark elements . . . . .	123
4.2.6	The state of the art of subdivision-based IgA . . . . .	125
4.3	The boundary element method (BEM) . . . . .	126
4.3.1	The boundary integral representation for 2D Laplace . . . . .	127
4.3.2	Discretisation . . . . .	128
4.3.3	Quadrature . . . . .	130
4.3.4	BEM and IgA . . . . .	132
4.4	Spline-enhanced methods . . . . .	132
4.4.1	Subdivision-enhanced FEM . . . . .	135
5	<b>Rendering aspects of curves and surfaces</b> . . . . .	<b>138</b>
5.1	Graphics APIs . . . . .	140
5.1.1	OpenGL . . . . .	140
5.1.2	Alternatives . . . . .	141
5.1.3	GPGPU . . . . .	141
5.1.4	Web-based APIs . . . . .	143
5.2	Tessellation . . . . .	143
5.2.1	Tesellation for multi-sided patches . . . . .	145
5.3	Subdivision shading . . . . .	149
6	<b>Colour gradients in vector graphics</b> . . . . .	<b>156</b>
6.1	Primitives . . . . .	158
6.2	Gradient meshes . . . . .	158
6.2.1	Local refinement . . . . .	163
6.2.2	More flexible topology . . . . .	170
6.2.3	Sharp colour transitions . . . . .	173
6.2.4	Gallery . . . . .	174
6.2.5	Colour spaces . . . . .	174
6.2.6	Gradient meshes and the web . . . . .	175
6.3	Diffusion curves . . . . .	177

6.3.1	Solvers . . . . .	178
6.3.2	Improvements . . . . .	180
<b>7</b>	<b>Conclusion and future work</b>	<b>182</b>
7.1	Conclusion . . . . .	184
7.2	Future work . . . . .	185
<b>A</b>	<b>Local refinement</b>	<b>190</b>
A.1	Hierarchical B-splines . . . . .	190
A.2	Truncated hierarchical B-splines . . . . .	192
<b>B</b>	<b>Meshes</b>	<b>193</b>
B.1	Generating three-valent meshes . . . . .	193
B.2	A data structure for curve networks . . . . .	194
<b>C</b>	<b>Tensors</b>	<b>196</b>
C.1	Terminology and notation . . . . .	196
C.2	Triple products of subdivision splines . . . . .	197
	<b>Bibliography</b>	<b>198</b>
	<b>Acknowledgement</b>	<b>218</b>

# Preface

Autumn 2010, a Friday evening in September. I am on a train that just left Eindhoven and is headed towards Utrecht Centraal, where I will have to change trains to get to Ede-Wageningen. With a duffle bag full of laundry on the seat next to me, I am on my way to my parents for the weekend.

I look around for something to read for the remainder of the journey, and find a newspaper left behind by someone. Browsing through it, an article about a short upcoming computer-animated film catches my eye. The film is called *Sintel* and has apparently been created using only open-source software, in particular some software called BLENDER. I am intrigued and write down these names. Without smartphone or functioning WiFi in the compartment, I will have to wait until later that weekend to look for more information.

A couple of days later I have repeatedly watched the trailer for *Sintel* and have downloaded BLENDER. I am impressed but not quite sure how to actually create something with it. Eventually I stumble upon some online tutorials mentioning modifiers, in particular *subdivision surfaces*.

Fast-forwarding a couple of weeks, I am at one of my first group meetings of the research group that I recently joined to pursue my MSc degree in mechanical engineering. Today's speaker is Clemens Verhoosel, the lecturer of a course I am looking forward to. The talk focuses on *isogeometric analysis* (IgA), which aims at re-using a geometry's description to define a solution space used when running numerical analysis on it. I catch something about *splines* and *free-form shapes*, which reminds me of the subdivision surfaces in BLENDER. Afterwards I have a chat with Clemens and ask whether he is familiar with subdivision. Although he does not know the details, he has heard about it and eventually we decide to do a short project on 2D subdivision surfaces for IgA. Both curious and motivated by the preliminary results, we agree that a more extensive approach would make a challenging topic for an MSc thesis.

Slowly, my desk is filling up with piles of printed lecture notes on splines (something which has not changed since), a topic which I have come to appreciate greatly. So much in fact, that shortly after obtaining my MSc degree, I find myself at the University of Cambridge, researching subdivision surfaces at the department of computer science. Here I meet Malcolm Sabin, Jiří Kosinka and other people with whom I travel to my first workshops and conferences. Sharing an office with Jiří allows for daily discussions about spline-related things and matters that are not directly work-related. It is an altogether great experience to live and work abroad. Unfortunately, as the year progresses we learn that the subdivision project we are part of did not get the expected extension. We will have to leave.

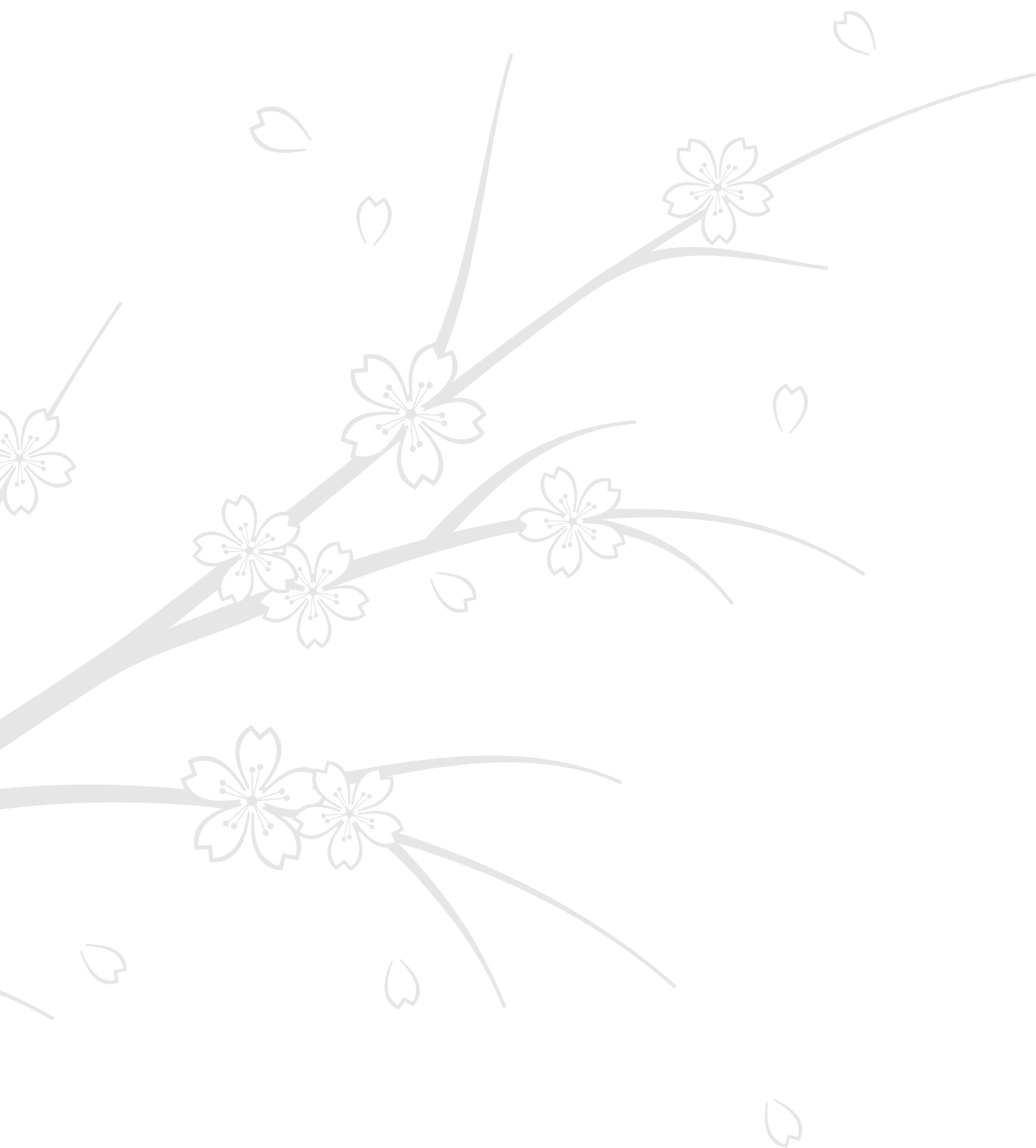
I end up in Leuven, working on an interesting topic for a while until I realise it is not the right environment for me to pursue a PhD. After long deliberation I decide to quit.

In the meantime, Jiří has found a position at the University of Groningen, where he will start next academic year as assistant professor. It turns out it comes with the position for a PhD candidate...



# 1 | Introduction





In this introductory chapter, I look ahead at the concepts and contributions discussed in the following chapters. In addition, I also comment on the writing style used throughout this dissertation.

And so, I started as a PhD candidate in Groningen — third time indeed proved to be the charm. Nevertheless, research-wise we initially settled on quite a different topic: the morphing of surfaces by means of using subdivision in the time domain<sup>†</sup>. Following a literature study on the massive research area of morphing, progress was unfortunately quite slow. Adding to that some personal matters early on in the second year, ultimately the only thing that morphed was the topic of my PhD — we decided to look into the use of splines for colour propagation in vector graphics instead, with some room for other projects.

Several publications and conference presentations later, I am now at the point of finishing my PhD and the accompanying dissertation. The writing process has been an insightful as well as enjoyable phase of the project, allowing me to spot new connections and directions for future research. As will become clear later on, at least two follow-up publications are expected to appear in the near future. Regarding these, preliminary results are included, though the eventual conclusions will naturally have to wait.

## 1.1 Content and contributions

Without further ado, let me introduce the main characters of this dissertation — the splines. They appear in many forms and bring such richness and elegance, that at times I feel honoured to study them (and even better, make my living by doing so). As they are involved in most parts of my research, it seems logical to include a chapter focused on some of their basic theory. Constructions for selected spline curves and surfaces, along with several useful techniques, are discussed in Chapter 2.

Following these spline basics, I then turn to bivariate splines of arbitrary manifold topology. Although this specialisation is quite a mouthful, it is actually an intuitive topic full of beautiful mathematics. The motivation here is twofold. First, composite  $G^1$  surfaces of arbitrary manifold topology are required in the context of vector graphics for the study of flexible gradient meshes. Secondly, one of the side projects considers a completed spline-based bivariate subdivision scheme aimed at the subdivision of three-valent (think mostly hexagonal) meshes that results in an overall  $C^1$  surface composed of triangular spline patches. These topics are treated in Chapter 3.

Next, I temporarily return to my mechanical engineering roots by considering various spline-based numerical methods. With the finite element method (FEM) — in a spline context nowadays often referred to as isogeometric analysis — as the most prominent one, I take the opportunity to highlight its main facets using a basic example, the Poisson equation. There are many interesting aspects of FEM, among those numerical integration and local refinement. The former is discussed in some detail, as it is the main ingredient for a collaboration on improved quadrature for subdivision splines. The latter is kept to a minimum, with spline refinement discussed in an appendix. Following FEM, I take a brief look at the boundary element method (BEM), which somehow is still a somewhat underexposed topic. The main reason for including it is its use in vector graphics for a primitive known as the diffusion curve. Finally, the spline-enhanced approach serves as a middle way between bivariate and trivariate meshing while still

---

<sup>†</sup>Thanks to Tom Cashman for sharing his idea with us!

enjoying the exact geometry. It is the topic of a collaboration I did with some of my former colleagues from Leuven. Chapter 4 discusses the three methods.

With the mathematical and engineering components present, it is then time to re-focus on computer science with a more in-depth look at various graphics APIs that are available nowadays, providing or facilitating tessellation and shading. These are the contents of Chapter 5.

Finally, I mix most of these topics, add some artistic flavouring, and look into vector graphics with an improved gradient mesh primitive and an overview of diffusion curves. This is discussed in the richly illustrated Chapter 6.

The dissertation is then wrapped up with an overall conclusion in Chapter 7, along with some thoughts on directions suitable for future research.

Summarising, my main contributions are as follows (in order of their discussion in this dissertation):

- A completed  $C^1$  bivariate subdivision scheme based on cubic half-box splines; see Chapter 3. It completes the list of low-degree (box) spline-based subdivision schemes and provides insight in the structure of control nets as well as ineffective eigenvectors.
- Improved quadrature rules for Catmull–Clark subdivision splines (follow-up publication expected); see Chapter 4. This allows for more efficient numerical simulations in the context of selected spline-based numerical methods.
- An improved gradient mesh primitive supporting local refinement and several other enhancements (follow-up publication expected); see Chapter 6. It facilitates the creation of resolution-independent (almost) photorealistic illustrations.

In addition to these, other contributions include:

- A spline-enhanced numerical method based on Catmull–Clark subdivision; see Chapter 4. It can be interpreted as a method in between a (classical) finite element approach for tetrahedral meshes and isogeometric analysis for subdivision solids.
- Various OpenGL tessellation strategies for multi-sided patches; see Chapter 5. This allows for efficient visualisation of multi-sided patches using the GPU.
- Improved subdivision shading; see Chapter 5. This incremental contribution allows to better fine-tune the shading of subdivision surfaces around extraordinary vertices.

## 1.2 About the writing style

The first text I wrote that was of a length longer than a typical report was my BSc thesis. It was a new experience, using both  $\text{\LaTeX}$  and INKSCAPE, and I enjoyed it quite a bit. As it was on a topic barely documented at that time, I decided to write it from a somewhat didactical point of view, and in addition provide a web-based tutorial on it. Over the years I have received quite a few positive emails about it, which inspired me to do something similar with future texts. In the case of my MSc thesis, I had to rush

it a little as the next position started earlier than the originally planned end date of the project. As such, only some chapters (including one on box splines) were written in a similar style.

Writing this dissertation has been another opportunity to hone my educational writing skills. Most of the text has been written in a somewhat colloquial form — I often use ‘we’ to refer to the reader and myself, though occasionally it refers to my co-authors. Furthermore, as I prefer intuitive and visual explanations over abstract mathematical ones (a good illustration is worth at least a paragraph, to slightly rephrase a common saying), formal proofs are rather scarce. Examples are added throughout the text in order to explain the various concepts introduced. As such, it has resulted in a style that is similar to the approach taken in textbooks on mathematical topics ‘for engineers’, which is often my default option for a first book on a subject unfamiliar to me. It also explains the title of this dissertation, which in some sense is also a wink to my background. Motivated by the positive experience of providing web-based content, I have decided to do a similar thing here — the plan is to provide lecture notes, interactive web-based applications and links to personal projects on GitHub on [SplinesForEngineers.com](https://SplinesForEngineers.com).

Ultimately, I believe that communication about one’s research is one of the fundamental aspects of a PhD project. With that in mind, why not do it in one’s own style?

## 1.3 About the illustrations

Last but not least, some words on the illustrations used throughout the dissertation. The one on the front cover depicts a mechanical spline, i.e. a thin, flexible strip of wood pinned down by weights known as spline ducks or whales. On the back, the images of the characteristic maps associated with the half-box spline subdivision scheme are shown for a couple of valencies. An example of the improved gradient mesh primitive adorns the bookmark/invitation.

Illustrations were created in vector format whenever possible, which should help to ensure a high-quality printed version, as well as provide a pleasant experience when zooming in onto the digital version.

Enjoy reading!

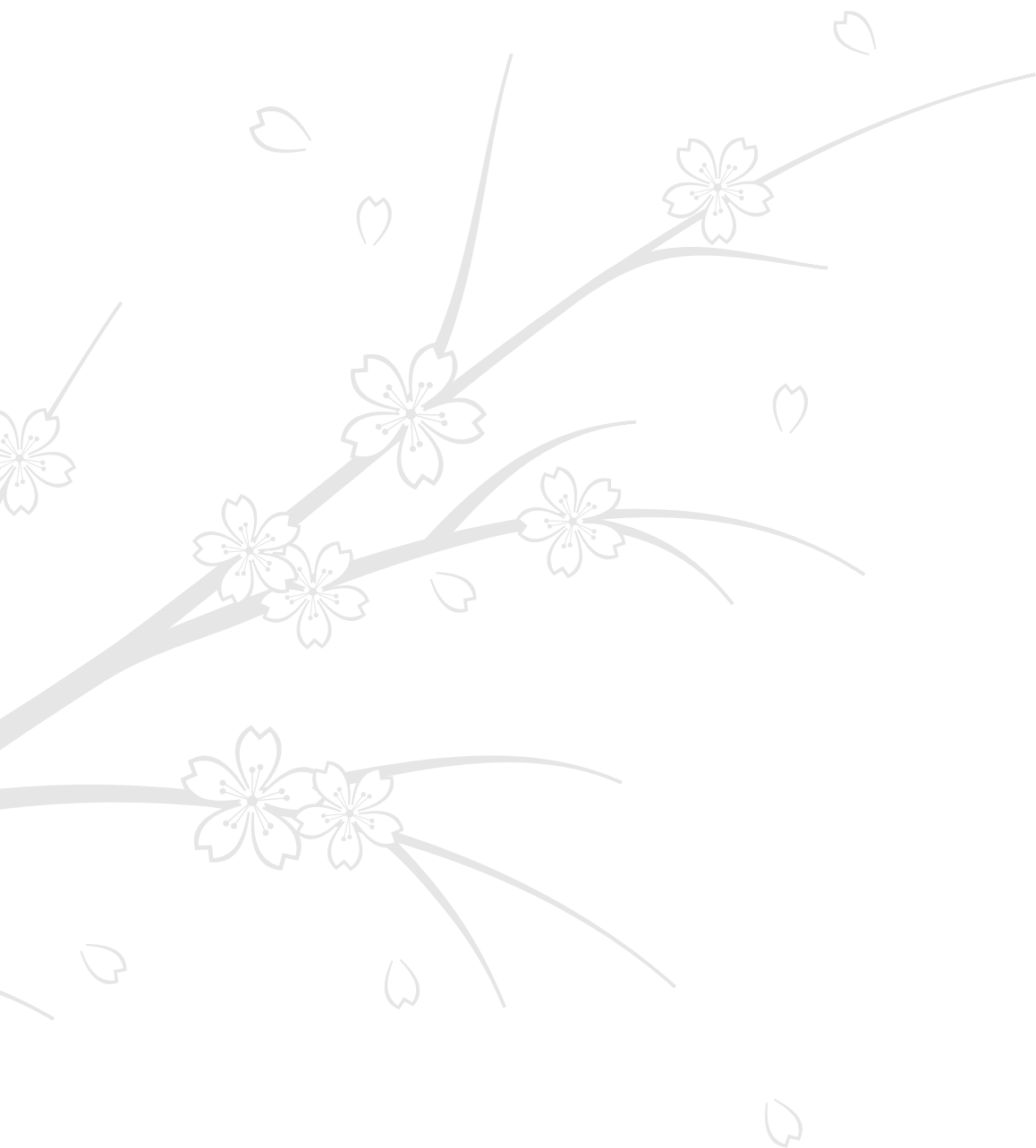
Groningen,  
October 2019

Pieter Barendrecht



## 2 | Spline basics





Splines form the foundation for the following chapters in this dissertation. As such, they deserve a preliminary chapter of their own. We discuss a selected set of commonly used spline curves and -patches and conclude with an overview on the refinement of splines.



## 2.1 Spline curves

We begin our journey in the world of splines by considering Bézier curves and B-spline curves, followed by a couple of alternative representations of polynomial spline curves. In the next section we generalise this univariate view to a bivariate one. For a more in-depth discussion of splines in general, see e.g. [HL93; Far02; FHK02; PBP02; Sal06].

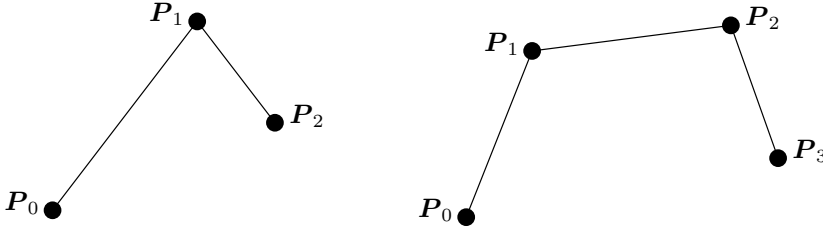
### 2.1.1 Bézier curves

Béziars are a natural place to start our exposition of splines in general, as they can be regarded as the building block for a large number of other spline types. Here, we use a geometric approach to introduce Bézier curves.

Given a sequence of  $d + 1$  points  $P_k \in \mathbb{R}^n$  with  $n \in \{2, 3\}$ , we are interested in using these points to define and control a parametric curve in  $\mathbb{R}^n$ . Our first step is to construct a *control polygon*<sup>†</sup> by connecting consecutive points  $P_k$  and  $P_{k+1}$  for  $k \in [0, d - 1] \subset \mathbb{Z}$ . The resulting edges between the points can be regarded as linear interpolations

$$P_k(t) = (1 - t)P_k + tP_{k+1}, \quad t \in [0, 1], \quad (2.1)$$

together resulting in a piecewise linear curve. Two examples are shown in Figure 2.1.



**Figure 2.1:** Two control polygons, linearly interpolating between control points  $P_k$ . On the left we have  $d = 2$ , on the right we have  $d = 3$ .

Our next step is to fix a value for  $t \in [0, 1]$  and evaluate all  $P_k(t)$  at this value, resulting in a set of  $d$  new points  $Q_k$  (see Figure 2.2). These points can again be connected by edges, yielding the linear interpolations

$$Q_k(t) = (1 - t)Q_k + tQ_{k+1}, \quad t \in [0, 1]. \quad (2.2)$$

Evaluation of the  $Q_k(t)$  at the previously fixed value of  $t$  then provides us with  $d - 1$  new points  $C_k$ .

In the case of  $d = 2$ , we are now left with a single point  $C_0$ . Substituting (2.1) in (2.2), with  $Q_k = P_k(t)$  and  $C_0 = Q_0(t)$  for non-fixed  $t \in [0, 1]$ , we obtain

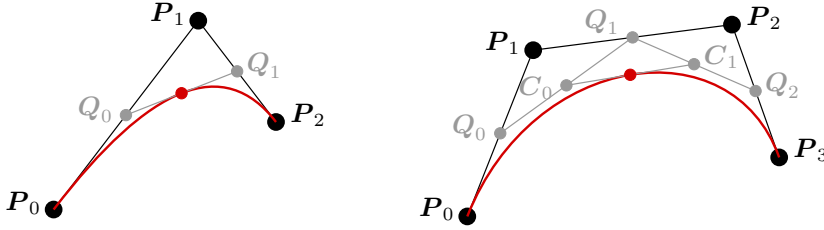
$$\begin{aligned} Q_0(t) &= (1 - t)P_0(t) + tP_1(t) \\ &= (1 - t)^2P_0 + 2t(1 - t)P_1 + t^2P_2, \end{aligned} \quad (2.3)$$

<sup>†</sup>Control *polyline* might be a better phrase here, but control polygon is the conventional term. Only for cyclic curves do we get a true polygon.

which is a *quadratic Bézier curve*. For  $d = 3$  and higher, we simply repeat the procedure until we are left with a single point. Taking  $d = 3$  as example, we linearly interpolate between  $C_0 = Q_0(t)$  and  $C_1 = Q_1(t)$  to obtain

$$\begin{aligned} C_0(t) &= (1-t)Q_0(t) + tQ_1(t) \\ &= (1-t)^3P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3, \end{aligned} \quad (2.4)$$

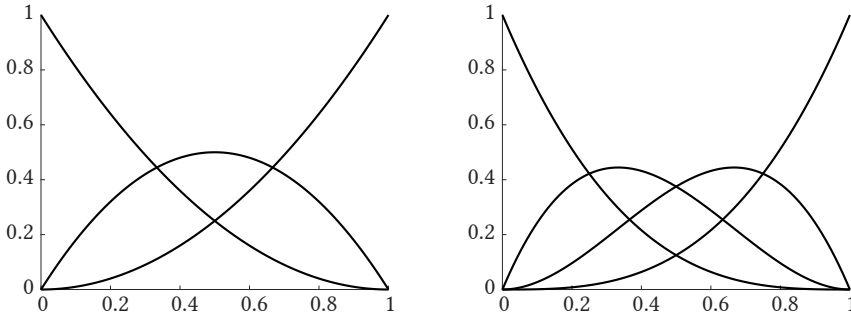
which is a *cubic Bézier curve*. The curves associated with the control polygons shown in Figure 2.1 are illustrated in Figure 2.2.



**Figure 2.2:** Quadratic Bézier curve (left) and cubic Bézier curve (right) corresponding to the control polygons from Figure 2.1.

The polynomial functions appearing in (2.3) and (2.4) are the *Bernstein polynomials*  $B_k(t)$  of degree  $d$ ; they are visualised in Figure 2.3. Clearly, these generalise to

$$B_k^d(t) = \binom{d}{k} (1-t)^{d-k} t^k. \quad (2.5)$$



**Figure 2.3:** The quadratic Bernstein polynomials (left) and cubic Bernstein polynomials (right).

It follows that (2.3) and (2.4) generalise to Bézier curves  $B(t)$  of arbitrary degree  $d$  (also referred to as order  $d + 1$ ):

$$B(t) = \sum_{k=0}^d B_k^d(t) P_k. \quad (2.6)$$

Note that the Bernsteins are — in this setting — a natural by-product of the geometric construction we used, commonly known as *de Casteljau's algorithm*. The  $\mathcal{B}_k^d(t)$  form a basis for the polynomials of degree  $d$ , partition unity, and enjoy several other desirable properties.

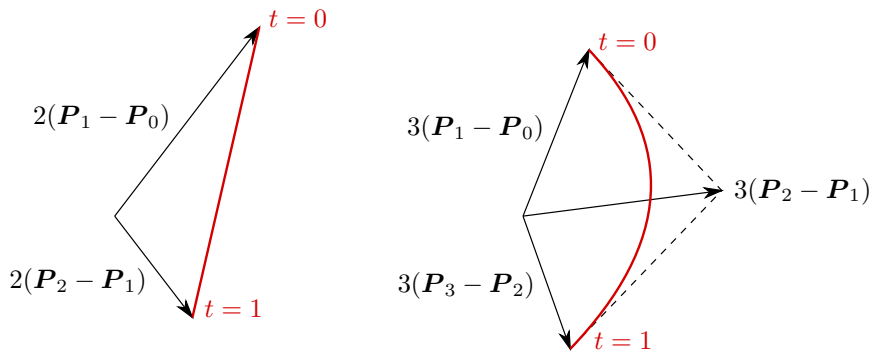
Before we study the behaviour of Bézier curves more closely, we first consider their derivatives. From (2.6) we see that  $B'(t) = \frac{dB(t)}{dt}$  follows by taking the derivatives of the Bernsteins. In turn, the derivative of (2.5) follows from the product rule — subsequently re-writing the result shows that

$$(\mathcal{B}_k^d)'(t) = d(\mathcal{B}_{k-1}^{d-1}(t) - \mathcal{B}_k^{d-1}(t)). \quad (2.7)$$

Note that for indices  $k < 0$  and  $k > d$  the  $\mathcal{B}_k^d(t)$  vanish. Substituting (2.7) into (2.6) then gives us (after re-ordering)

$$B'(t) = d \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(t)(P_{k+1} - P_k). \quad (2.8)$$

The derivatives of the curves shown in Figure 2.2 are plotted in Figure 2.4. The geometrical significance of these plots is that they visualise the *tangent vector* at any point  $t \in [0, 1]$  on the original curve, which makes them *hodographs*.



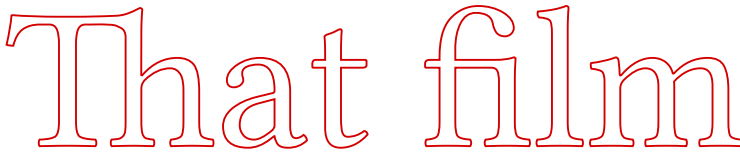
**Figure 2.4:** Derivatives of the quadratic (left) and cubic Bézier curve (right) from Figure 2.2.

From (2.5) and (2.6) we observe that for any degree  $d$  the first and last control point are interpolated by a Bézier curve, whereas the other points are generally not. Instead, they merely ‘pull’ the curve in their direction. Secondly, from (2.8) it follows that the tangent vectors at the start and end of the curve are simply the (scaled) vectors spanned by the first and second control point, or the penultimate and last point, respectively. Combining these two observations explains why the *cubic* Bézier curve is so popular in design — the *degrees of freedom* (DoFs) provided by the four control points allow the user to control the initial and final position of the curve, as well as the tangents at those positions. Curves of lower order do not provide enough DoFs to do so (e.g. for quadratic Béziars the two tangents cannot be set independently as the middle control point is involved in both). In contrast, curves of higher order provide more DoFs, though

these are not directly geometrically intuitive in their use (e.g. for quintic Béziers there are enough DoFs to also control the second derivative at the initial and final position).

Still, for the design of intricately curved shapes, a single cubic Bézier curve is often not sufficient. Instead of using higher order curves, the common approach is to use multiple cubic Béziers that are connected to one another, each pair sharing a control point. It is often the case that these connections are required to be *tangent-continuous*, meaning that the direction of the tangents on both sides of the connection should be the same. Such a connection is also referred to as  $G^1$ , which stands for *geometric continuity* of the first order. If also the magnitude of the tangents is the same, the connection is  $C^1$ , which stands for *parametric continuity* of the first derivative.

Fonts are a good example of composite Bézier design<sup>†</sup>, see Figure 2.5. Most modern (e.g. OpenType) fonts use cubic Béziers, whereas the older (e.g. TrueType) fonts are based on quadratic Béziers. Modern fonts have many interesting attributes<sup>‡</sup>, but unfortunately this is not the place to digress on this beautiful topic.



**Figure 2.5:** The outline of several glyphs of the Linux Libertine font, including the ligatures *Th* and *fi*.

We remark that quadratic Béziers can be interpreted as cubic ones by the process of *degree-elevation*. This is realised by multiplying (2.3) by  $1 = ((1-t) + t)$ , resulting in a sum of cubic Bernsteins each multiplied by a linear combination of two original control points, ultimately defining a cubic Bézier. In general, the resulting control points follow as

$$\mathbf{R}_k = \frac{k}{d} \mathbf{P}_{k-1} + \frac{d-k}{d} \mathbf{P}_k, \quad (2.9)$$

where  $d$  refers to the degree of the degree-elevated curve.

Realising a  $G^1$  connection between two Béziers is straightforward, and results in three co-linear control points in the control polygon. The additional constraint for a  $C^1$  connection is that these three points should be equidistant. For two cubic Bézier curves – sharing  $\mathbf{P}_3$  such that they are  $C^0$  continuous – to connect  $C^1$ , we thus have  $\mathbf{P}_4 = \mathbf{P}_3 + (\mathbf{P}_3 - \mathbf{P}_2)$ . Note that we can repeatedly apply (2.8), so that a  $C^2$  connection follows by ensuring the previous conditions and subsequently setting  $\mathbf{P}_5 = \mathbf{P}_1 + 2(\mathbf{P}_4 - \mathbf{P}_2)$ . Likewise, for the two segments to be  $C^3$ , it follows that  $\mathbf{P}_6 = \mathbf{P}_3 + (\mathbf{P}_3 - \mathbf{P}_0) + 3(\mathbf{P}_1 - \mathbf{P}_2) + 3(\mathbf{P}_5 - \mathbf{P}_4)$ . In this last case, the second segment is actually a parametric continuation of the first segment.

<sup>†</sup> Spiro splines [Lev09] are an interesting alternative for font design and are available in FONTFORGE and INKSCAPE.

<sup>‡</sup> Two of these attributes are  *Kerning* and *ligatures*, which are both used extensively by X<sub>2</sub>L<sub>A</sub>T<sub>E</sub>X. Spotting ligatures in a text is an enjoyable game and often provides a reasonable indication of what a font has to offer!

The above might raise the question whether composite curves connecting with e.g.  $C^1$  continuity can be represented more efficiently, as the middle of these three co-linear points is merely a convex combination of the other two. The answer is positive and can be obtained by studying B-spline curves.

### 2.1.2 B-spline curves

B-splines<sup>†</sup> come in many different flavours, can be defined using a variety of ways, and form an extensive area of research. Merely interpreting them as an efficient way to represent composite Béziers does perhaps not do them justice, but as this is the direction we come from, it is our starting point.

An important aspect to consider when connecting Bézier curves is whether all segments should be defined on parameter intervals of the same length. If we choose to do so, and set that interval to be of unit length, we obtain a vector of parameter values  $\Xi = [0, 1, 2, \dots, m]$ , where  $m$  indicates the number of segments. These parameter values indicate where our segments are connected, or *tied together*, in parameter space. For this reason, they are often referred to as *knots*, and  $\Xi$  as the *knot-vector*.

Let us consider a composite quadratic Bézier curve of two segments that connect with  $C^1$  continuity. We thus have control points  $\mathbf{P}_0, \dots, \mathbf{P}_4$ , with as corresponding basis functions  $M_0(t), \dots, M_4(t)$  the Bernsteins, which for the second curve segment are shifted by one unit:

$$\begin{aligned} M_0(t) &= (1-t)^2 & \text{for } t \in [0, 1], \\ M_1(t) &= 2t(1-t) & \text{for } t \in [0, 1], \\ M_2(t) &= \begin{cases} t^2 & \text{for } t \in [0, 1], \\ (2-t)^2 & \text{for } t \in [1, 2], \end{cases} \\ M_3(t) &= 2(t-1)(2-t) & \text{for } t \in [1, 2], \\ M_4(t) &= (t-1)^2 & \text{for } t \in [1, 2]. \end{aligned}$$

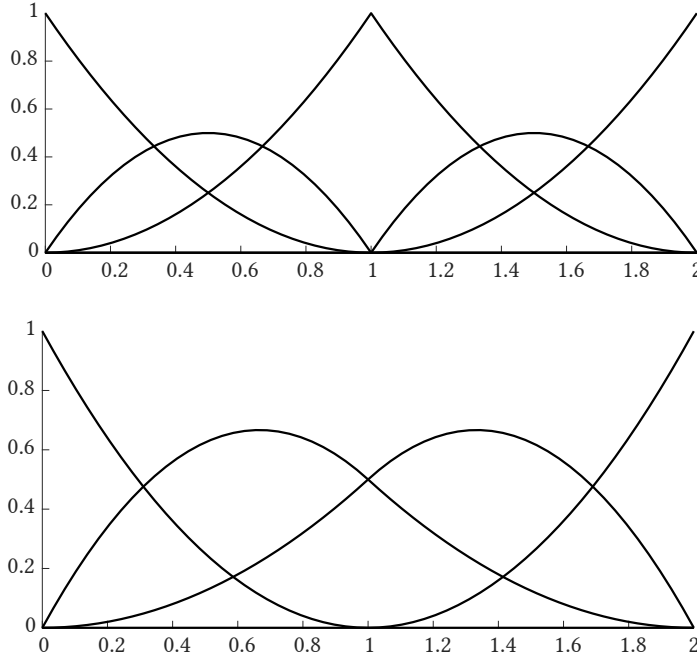
These basis functions are visualised in Figure 2.6. The composite curve can now be expressed as  $C(t) = \sum_{k=0}^4 M_k(t) \mathbf{P}_k$ . Using  $\mathbf{P}_2 = \frac{1}{2} \mathbf{P}_1 + \frac{1}{2} \mathbf{P}_3$ , which holds because of the  $C^1$  connectivity, we can re-write the expression as

$$\begin{aligned} C(t) &= M_0(t) \mathbf{P}_0 + M_1(t) \mathbf{P}_1 + M_2(t) \left( \frac{1}{2} \mathbf{P}_1 + \frac{1}{2} \mathbf{P}_3 \right) + M_3(t) \mathbf{P}_3 + M_4(t) \mathbf{P}_4 \\ &= M_0(t) \mathbf{P}_0 + \left( M_1(t) + \frac{1}{2} M_2(t) \right) \mathbf{P}_1 + \left( \frac{1}{2} M_2(t) + M_3(t) \right) \mathbf{P}_3 + M_4(t) \mathbf{P}_4. \end{aligned}$$

This shows that we can omit the middle control point in a set of three co-linear points, but only if we update the basis accordingly (see Figure 2.6). The result is that  $C^1$  continuity is now hard-coded in the basis.

For a composite  $C^1$  quadratic curve with  $m$  segments we can apply the same procedure, substituting  $\mathbf{P}_k = \frac{1}{2} \mathbf{P}_{k-1} + \frac{1}{2} \mathbf{P}_{k+1}$  for all even  $k : 0 < k < 2m$ . The control points between two omitted points then become associated with basis functions that

<sup>†</sup>The term B-spline means *basic spline* and usually refers to a basis function. B-spline curve refers to a parametric curve defined using B-splines, though in practice the term B-spline is often used for both.



**Figure 2.6:** Two bases (spanning different spaces) for a composite quadratic curve. At the top we have the common Bernsteins, at the bottom we have the modified basis with built-in  $C^1$  continuity of the connection.

are composed of three parts. For example, for  $m = 3$  the basis function associated with  $P_3$  is

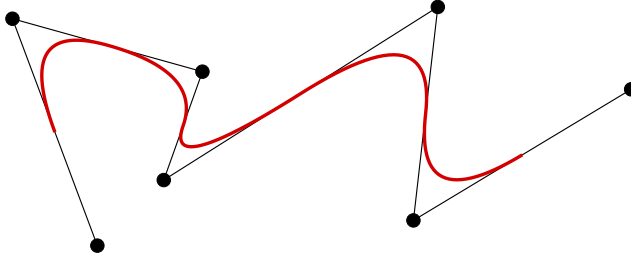
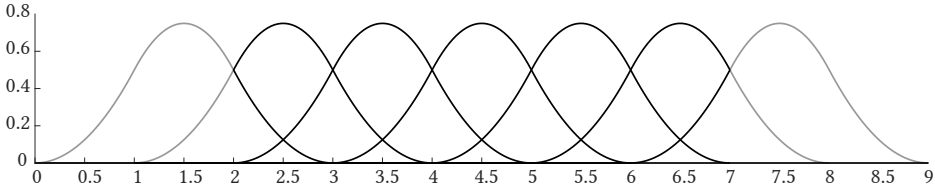
$$\overline{M}_3(t) = \frac{1}{2}M_2(t) + M_3(t) + \frac{1}{2}M_4(t),$$

which equals

$$\overline{M}_3(t) = \begin{cases} \frac{1}{2}t^2 & \text{for } t \in [0, 1], \\ \frac{1}{2}(2-t)^2 + 2(t-1)(2-t) + \frac{1}{2}(t-1)^2 & \text{for } t \in [1, 2], \\ \frac{1}{2}(3-t)^2 & \text{for } t \in [2, 3]. \end{cases} \quad (2.10)$$

This basis function is the *uniform quadratic B-spline*, where uniform refers to the equally spaced parameter values (knots) in the knot-vector.

For  $m$  large enough,  $\overline{M}_5(t)$ ,  $\overline{M}_7(t)$  and so on, are simply  $\overline{M}_3(t)$  shifted by one or more units. In the extreme case, we can define a basis for a composite quadratic curve solely as a sequence of shifted versions of the uniform quadratic B-spline. Like quadratic Béziers, we still need three control points (and their associated basis functions) to define the first segment, but unlike composite quadratic Béziers, every additional control point now provides us with an additional segment that connects with  $C^1$  continuity to the previous segment. A drawback is that the first and last points are no longer interpolated. An example is shown in Figure 2.7.



**Figure 2.7:** Uniform quadratic B-splines with the region satisfying a partition of unity highlighted in black (top) and a corresponding quadratic B-spline curve (bottom).

Similar procedures can be applied to obtain uniform B-splines (UBS) of any order<sup>†</sup>, though other — more straightforward — methods to construct uniform B-splines exist. To illustrate this, let us consider uniform cubic B-spline curves, which are composed of cubic Béziers connecting with  $C^2$  continuity. From the quadratic case, we know that  $P_3 = \frac{1}{2}P_2 + \frac{1}{2}P_4$ , though merely using this fact would result in a cubic  $C^1$  basis. Instead, the observation is that we have  $P_2 + (P_2 - P_1) = P_4 + (P_4 - P_5)$ . If we label this position as a new point  $I$ , it follows that  $P_2$  and  $P_4$  (and therefore also  $P_3$ ) can be obtained from  $P_1$ ,  $I$  and  $P_5$ , which allows the construction of a basis with built-in  $C^2$  continuity.

The notion of uniform B-splines can be generalised to *non-uniform* B-splines (NUBS), which can be constructed starting from a non-uniform knot-vector  $\Xi$  (i.e. the knots are no longer equidistant in this case).

Furthermore, NUBS can be further extended to NURBS [Far91; PT97; Rog00], which allows the designer to assign scalar weights to control points<sup>‡</sup> in order to specify how much the curve should be attracted to these points. The control points are multiplied by these weights, which means that the basis functions have to be modified in order to still partition unity. The result is a set of *rational* basis functions, which explains the R in NURBS. They are popular in certain areas of design as — in contrast to NUBS — NURBS can represent conic sections such as circular arcs exactly.

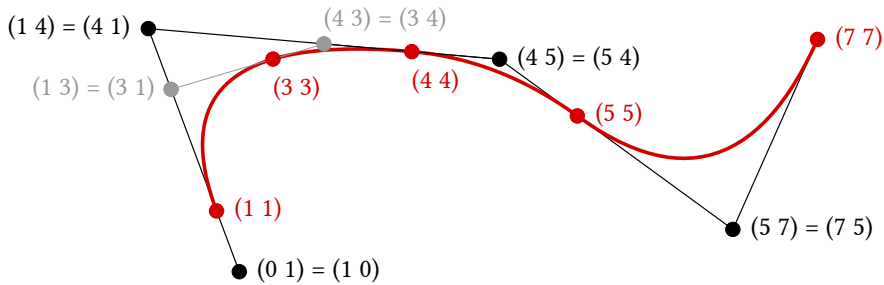
The concept of knot-vectors can be generalised even further to contain duplicate knots, which causes the basis to be parametrically continuous only up to a certain derivative (e.g. non-uniform cubic B-splines with double knots are  $C^1$  instead of  $C^2$ ).

Although a detailed discussion of general non-uniform (rational) B-splines and their evaluation (requiring a derivation of the Cox-de Boor algorithm [Cox72; Boo72]) is

<sup>†</sup>Note that uniform B-splines of degree  $d$  are not merely  $C^1$ , but  $C^{d-1}$  continuous.

<sup>‡</sup>Using the concept of *homogeneous coordinates*, these scalar weights can be interpreted as an additional coordinate of the control points.

somewhat out of scope, the notion of *blossoming* [Gol02; Man06] – also referred to as polar forms [Ram89; Sei93] – should be mentioned, albeit only from a pragmatic point of view. The idea is to assign  $d$  consecutive knots from the knot-vector (referred to as a *blossom label*) to each control point in the control polygon of a general degree- $d$  B-spline curve. The  $d$  values in a blossom label can be re-ordered in any permutation. Affine interpolation can be applied between any two blossom labels with  $d-1$  matching knots. Finally, when all  $d$  values in a blossom label are identical, a point on the curve is reached (i.e. the image of the parameter value repeated in the blossom label). It follows that blossoming is a generalisation of de Casteljau’s algorithm from Béziers to B-splines; Figure 2.8 shows an example.



**Figure 2.8:** Blossoming in action on a non-uniform quadratic B-spline with knot-vector  $\Xi = [0, 0, 1, 4, 5, 7, 7, 7]$ . Note that the first and last knot are ignored in blossoming.

We conclude this section with an alternative definition of uniform B-splines, which will be helpful for both the development of bivariate splines and certain refinement relations later in this chapter. The approach is based on the convolution of two univariate functions.

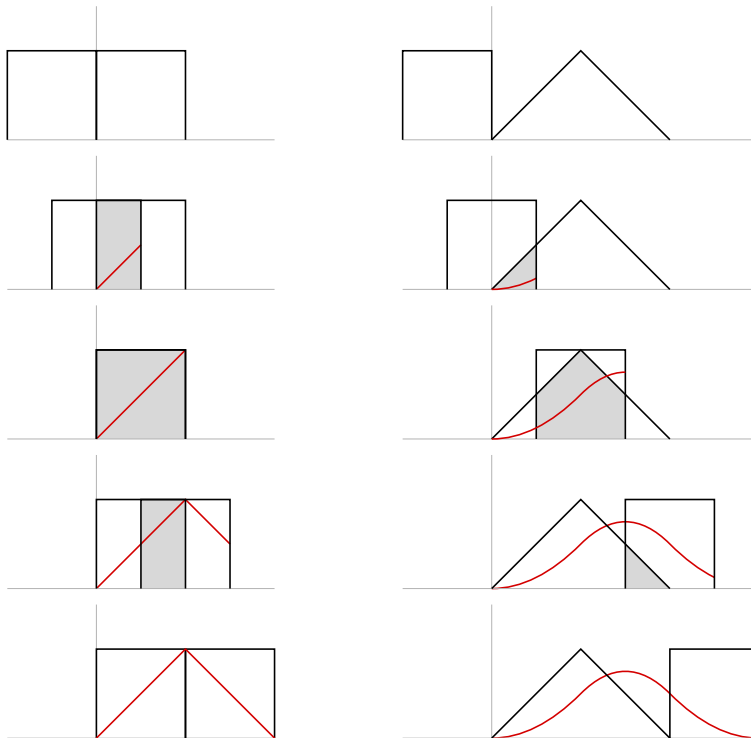
We first convolve the unit pulse  $f(t)$  (also known as the rectangular function or box(car) function) with itself. This results in the hat function  $g(t)$ , which is a piecewise linear function:  $g(t) = t$  for  $t \in [0, 1]$  and  $g(t) = (2 - t)$  for  $t \in [1, 2]$ . The process is illustrated in Figure 2.9 (left). Next, we convolve the unit pulse  $f(t)$  with the hat function  $g(t)$ . This involves reflecting  $f(\tau)$  to  $f(-\tau)$ , shifting the reflected  $f(t - \tau)$  for  $t \in [0, 3]$ , and integrating the overlap with  $g(\tau)$ . Note that in general, the convolution integrand is the *product* of  $f(t - \tau)$  and  $g(\tau)$ , but as  $f$  is the unit pulse in our case, the



geometric interpretation is to integrate the *overlap* of the two functions:

$$\begin{aligned} \int_0^t 1 \cdot \tau \, d\tau &= \left. \frac{1}{2} \tau^2 \right|_0^t = \frac{1}{2} t^2 && \text{for } t \in [0, 1], \\ \int_{t-1}^t 1 \cdot g(\tau) \, d\tau &= \int_{t-1}^1 1 \cdot \tau \, d\tau + \int_1^t 1 \cdot (2 - \tau) \, d\tau \\ &= \left. \frac{1}{2} \tau^2 \right|_{t-1}^1 + 2\tau - \left. \frac{1}{2} \tau^2 \right|_1^t \\ &= \left( -\frac{1}{2} t^2 + t \right) + \left( -\frac{1}{2} t^2 + 2t - 1.5 \right) \\ &= -t^2 + 3t - 1.5 && \text{for } t \in [1, 2], \\ \int_{t-1}^2 1 \cdot (2 - \tau) \, d\tau &= 2\tau - \left. \frac{1}{2} \tau^2 \right|_{t-1}^2 = \frac{1}{2} t^2 - 3t + 4.5 && \text{for } t \in [2, 3]. \end{aligned}$$

The result is the uniform quadratic B-spline, see Figure 2.9 (right). Additional steps of convolution with the unit pulse result in uniform B-splines of higher order.



**Figure 2.9:** Convolution of the unit pulse with itself (left) and convolution of the unit pulse with the hat function (right). The resulting functions are the hat function (uniform linear B-spline) and the uniform quadratic B-spline (shown in red), respectively.

Although swapping the roles of  $f(t)$  and  $g(t)$  in the convolution does not change

the outcome, i.e.  $(f * g)(t) = (g * f)(t)$ , it turns out to be a somewhat more convenient choice from a notational point of view. The reason is that in the case of  $(g * f)(t)$  the overlap always occurs within  $\tau \in [0, 1]$ , also for higher-order B-splines. In general, we therefore have

$$\mathcal{M}^{d+1}(t) = \int_0^1 \mathcal{M}^d(t - \tau) d\tau, \quad (2.11)$$

with  $\mathcal{M}^d(t)$  the uniform B-spline of degree  $d$ . We can use this definition to derive an expression for the derivative of uniform B-splines, which will be useful later:

$$\begin{aligned} \frac{d}{dt} \mathcal{M}^{d+1}(t) &= \frac{d}{dt} \int_0^1 \mathcal{M}^d(t - \tau) d\tau \\ &= - \int_0^1 \frac{d}{d\tau} \mathcal{M}^d(t - \tau) d\tau \\ &= - \left( \mathcal{M}^d(t - \tau) \Big|_0^1 \right) \\ &= \mathcal{M}^d(t) - \mathcal{M}^d(t - 1). \end{aligned} \quad (2.12)$$

### 2.1.3 Alternative representations

Apart from Béziers, there are several other ways to represent parametric curves. In this section we look at two of these alternative representations, *Lagrange* and *Hermite* curves. We will encounter their bivariate counterparts later on.

#### Lagrange curves

The rationale behind Lagrange curves is that they interpolate all control points of a control polygon. In certain settings this can be advantageous over Bézier curves, though the approach can also have its drawbacks.

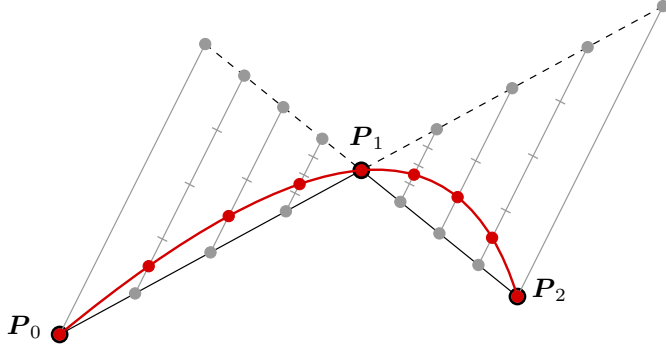
The first step in the construction of a Lagrange curve is to associate the control points with parameter values  $t_i$  at which they should be interpolated. Choosing the unit domain<sup>†</sup>  $t \in [0, 1]$ , a natural choice is to associate the control points with equidistant parameter values. That is, for a quadratic curve, those values are  $(t_0, t_1, t_2) = (0, \frac{1}{2}, 1)$ , whereas for cubics they are  $(t_0, t_1, t_2, t_3) = (0, \frac{1}{3}, \frac{2}{3}, 1)$ .

Taking the quadratic case as example, we can now use a *de Casteljau*-like approach to construct the curve. However, for this to work properly, we have to modify the control polygon. After all, its first edge  $\mathbf{P}_0 - \mathbf{P}_1$  is associated with  $t \in [0, \frac{1}{2}]$ , whereas the second edge  $\mathbf{P}_1 - \mathbf{P}_2$  is associated with  $t \in [\frac{1}{2}, 1]$ . The idea is to extend both edges such that they are associated with  $t \in [0, 1]$ , see Figure 2.10.

Applying de Casteljau's algorithm, we obtain

$$\begin{aligned} L_0(t) &= (1 - t)\mathbf{P}_0 + t(\mathbf{P}_1 + (\mathbf{P}_1 - \mathbf{P}_0)), \\ L_1(t) &= (1 - t)(\mathbf{P}_1 + (\mathbf{P}_1 - \mathbf{P}_2)) + t\mathbf{P}_2. \end{aligned}$$

<sup>†</sup>Note that choosing the *bi-unit* domain  $t \in [-1, 1]$  is a more common choice for Lagrange curves.

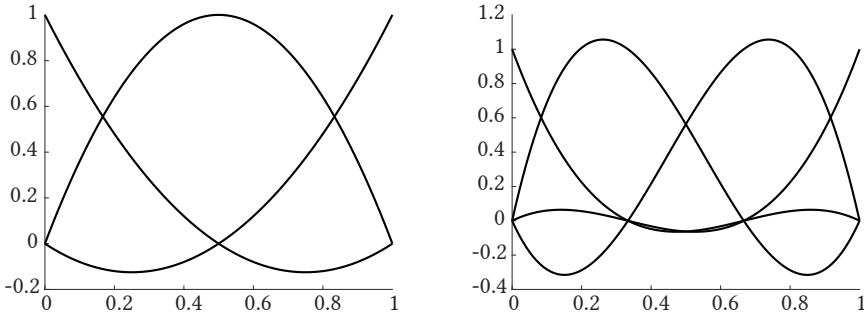


**Figure 2.10:** Construction of a quadratic Lagrange curve using de Casteljau's algorithm on a modified control polygon.

Evaluating  $L_0(t)$  and  $L_1(t)$  at the same value of  $t \in [0, 1]$  provides us with two points  $Q_0$  and  $Q_1$ . Next, we interpolate these two points and again evaluate at the same value of  $t$  to obtain

$$\begin{aligned} Q(t) &= (1-t)Q_0 + tQ_1 \\ &= (1-t)\left((1-t)P_0 + t(P_1 + (P_1 - P_0))\right) + t\left((1-t)(P_1 + (P_1 - P_2)) + tP_2\right) \\ &= \left((1-t)^2 - t(1-t)\right)P_0 + 4t(1-t)P_1 + \left(t^2 - t(1-t)\right)P_2 \\ &= \mathcal{L}_0^2(t)P_0 + \mathcal{L}_1^2(t)P_1 + \mathcal{L}_2^2(t)P_2. \end{aligned}$$

The basis functions  $\mathcal{L}_k^2(t)$ , known as the quadratic Lagrange polynomials, are plotted in Figure 2.11.



**Figure 2.11:** Quadratic Lagrange basis functions (left) and cubic Lagrange basis functions (right).

An important observation is that  $\mathcal{L}_k^2(t_l) = \delta_{kl}$ , where  $\delta_{kl}$  is the Kronecker delta. This holds for Lagrange polynomials of any order and can be used to construct them in a more straightforward manner. For example, the cubic Lagrange polynomial  $\mathcal{L}_0^3(t)$  now simply follows by initially setting it to  $(t-t_1)(t-t_2)(t-t_3) = (t-\frac{1}{3})(t-\frac{2}{3})(t-1)$

and subsequently scaling it<sup>†</sup> by  $1/(t_0 - t_1)(t_0 - t_2)(t_0 - t_3)$  to ensure that  $\mathcal{L}_0^3(t_0) = 1$ . In general, for degree  $d$  Lagrange polynomials we obtain

$$\mathcal{L}_k^d(t) = \prod_{\substack{l=0 \\ l \neq k}}^d \frac{t - t_l}{t_k - t_l}. \quad (2.13)$$

Partition of unity of the Lagrange polynomials directly follows from the Kronecker delta property – summing the  $d + 1$  basis functions yields a degree  $d$  polynomial that attains the value 1 at  $d + 1$  different parameter values.

A well-known drawback of higher-order Lagrange curves is that they often suffer from *Runge's phenomenon*, meaning that they exhibit undesired oscillatory behaviour.

### Hermite curves

The idea here is to construct curves  $C(t)$  on  $t \in [0, 1]$  that satisfy positional and tangential (and possibly higher order) data at  $t = 0$  and  $t = 1$ . As such, Hermite curves can only be defined for odd degrees<sup>‡</sup>  $d \geq 3$ .

Taking  $d = 3$  as an example, the aim is to define basis functions such that  $C(0) = \mathbf{P}_0$ ,  $C'(0) = \mathbf{T}_0$ ,  $C'(1) = \mathbf{T}_1$  and  $C(1) = \mathbf{P}_1$ , with  $\mathbf{P}_k$  and  $\mathbf{T}_k$  given positions and tangent vectors. Setting  $C(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3$  with unknown  $\mathbf{a}_l$ , we obtain

$$\begin{aligned} C(0) &= \mathbf{a}_0 = \mathbf{P}_0, \\ C'(0) &= \mathbf{a}_1 = \mathbf{T}_0, \\ C'(1) &= \mathbf{a}_1 + 2\mathbf{a}_2 + 3\mathbf{a}_3 = \mathbf{T}_1, \\ C(1) &= \mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3 = \mathbf{P}_1. \end{aligned}$$

This leaves us with two equations for  $\mathbf{a}_2$  and  $\mathbf{a}_3$ ,

$$\begin{aligned} 2\mathbf{a}_2 + 3\mathbf{a}_3 &= \mathbf{T}_1 - \mathbf{T}_0, \\ \mathbf{a}_2 + \mathbf{a}_3 &= \mathbf{P}_1 - \mathbf{P}_0 - \mathbf{T}_0, \end{aligned}$$

resulting in  $\mathbf{a}_2 = 3(\mathbf{P}_1 - \mathbf{P}_0) - 2\mathbf{T}_0 - \mathbf{T}_1$  and  $\mathbf{a}_3 = -2(\mathbf{P}_1 - \mathbf{P}_0) + \mathbf{T}_0 + \mathbf{T}_1$ . Substituting this in  $C(t)$  and re-arranging yields

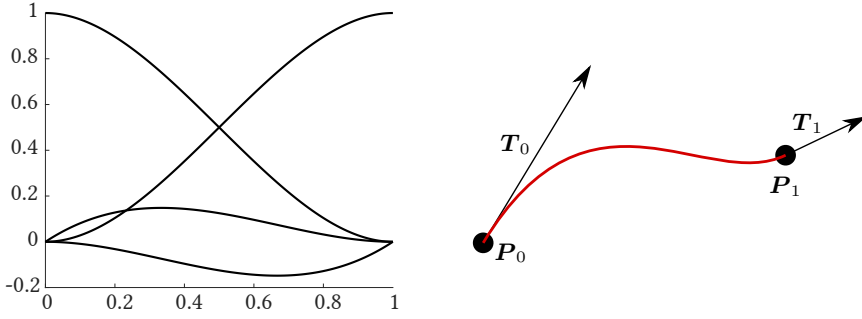
$$\begin{aligned} C(t) &= (1 - 3t^2 + 2t^3)\mathbf{P}_0 + (t - 2t^2 + t^3)\mathbf{T}_0 + (-t^2 + t^3)\mathbf{T}_1 + (3t^2 - 2t^3)\mathbf{P}_1 \\ &= \mathcal{H}_0^3(t)\mathbf{P}_0 + \mathcal{H}_1^3(t)\mathbf{T}_0 + \mathcal{H}_2^3(t)\mathbf{T}_1 + \mathcal{H}_3^3(t)\mathbf{P}_1, \end{aligned} \quad (2.14)$$

where  $\mathcal{H}_l^3(t)$  are the cubic Hermite basis functions (see Figure 2.12).

The same method can be followed to obtain a quintic Hermite basis (also satisfying given second derivative data at  $t = 0$  and  $t = 1$ ) as well as higher-order Hermite bases.

<sup>†</sup>The scaling factor is obtained by evaluating the initial polynomial at the relevant  $t_l$  and taking the reciprocal of it.

<sup>‡</sup>Not to be confused with *Hermite polynomials*, which form orthogonal bases for spaces of any dimension.



**Figure 2.12:** Cubic Hermite basis functions  $\mathcal{H}_l^3(t)$  (left) and a cubic Hermite curve, also referred to as a Ferguson cubic (right).

## 2.2 Spline patches

In this section we extend the ideas introduced in the previous section to the bivariate setting. We follow a similar order and therefore start with tensor-product and triangular Bézier patches, followed by a selected set of *polyhedral splines*. We also consider several alternative representations.

### 2.2.1 Bézier patches

The definition of Bézier curves can be directly extended to construct *tensor-product* patches. Where the curves are images of the unit interval  $t \in [0, 1]$ , the tensor-product patches are images of the unit square  $(u, v) \in [0, 1]^2$ . Given a  $(d+1) \times (d+1)$  *control net* or grid<sup>†</sup> of control points  $P_{kl}$ , we can construct  $d+1$  Bézier curves in either parametric direction. Assuming we construct them in the  $u$ -direction, evaluating all curves at the same value of  $u$  then results in  $d+1$  new points that serve as the control points for a Bézier curve in the other parametric direction, in our case the  $v$ -direction. Any point evaluated at  $v \in [0, 1]$  on this curve lies on the surface patch  $S(u, v)$ . Summarising, we have

$$S(u, v) = \sum_{l=0}^d \mathcal{B}_l^d(v) \left( \sum_{k=0}^d \mathcal{B}_k^d(u) P_{kl} \right), \quad (2.15)$$

which we can represent in *matrix form* as

$$S(u, v) = \begin{pmatrix} \mathcal{B}_0^d(u) & \mathcal{B}_1^d(u) & \dots & \mathcal{B}_d^d(u) \end{pmatrix} \begin{pmatrix} P_{00} & P_{01} & \dots & P_{0,d} \\ P_{10} & P_{11} & \dots & P_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ P_{d,0} & P_{d,1} & \dots & P_{d,d} \end{pmatrix} \begin{pmatrix} \mathcal{B}_0^d(v) \\ \mathcal{B}_1^d(v) \\ \vdots \\ \mathcal{B}_d^d(v) \end{pmatrix}.$$

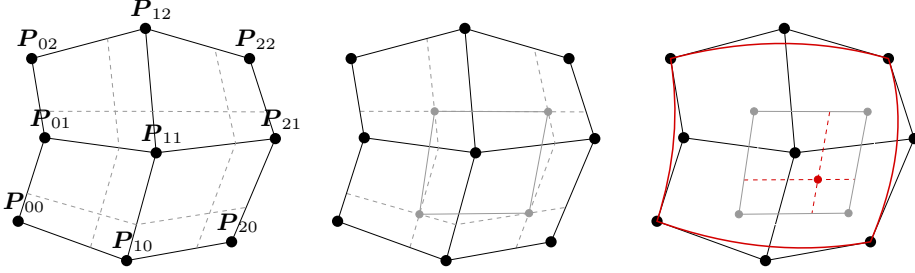
Note that we can also rewrite (2.15) into a single summation over  $(d+1) \times (d+1)$  tensor-product Bernsteins multiplied by the control points.

<sup>†</sup>The grid does not have to be square, but instead can be rectangular, resulting in a patch of different degrees in  $u$  and  $v$ .

An alternative method to evaluate a point on the patch is by extending de Casteljau's algorithm to the bivariate case. Neighbouring control points forming quads in the control net are interpolated bilinearly for fixed  $(u, v)$  values, resulting in new points

$$(1 - v) \left( (1 - u) \mathbf{P}_{kl} + u \mathbf{P}_{k+1,l} \right) + v \left( (1 - u) \mathbf{P}_{k,l+1} + u \mathbf{P}_{k+1,l+1} \right).$$

Note that these points form a  $d \times d$  grid. Similar to the univariate algorithm, the procedure is repeated until the grid consists of a single point, which is the point  $S(u, v)$ . Figure 2.13 illustrates the approach.



**Figure 2.13:** Using the bivariate version of de Casteljau's algorithm to evaluate the point corresponding to  $(u, v) = (\frac{2}{3}, \frac{1}{3})$  on a biquadratic Bézier patch. The boundary curves are also shown (right).

In addition to tensor-product Bézier patches, it is also possible to define *triangular Bézier patches*. A fundamental aspect in the definition of such patches are the *barycentric coordinates*, which are defined as follows. Given a triangular domain with corners  $A, B$  and  $C$ , any point  $p$  in the triangle can be represented as a linear combination  $p = uA + vB + wC$ . In other words,  $u, v$  and  $w$  are the barycentric coordinates of the point  $p$ , and are defined as ratios of areas  $\mathcal{A}()$  of triangles<sup>†</sup>:

$$u = \frac{\mathcal{A}(\triangle pBC)}{\mathcal{A}(\triangle ABC)}, \quad v = \frac{\mathcal{A}(\triangle pCA)}{\mathcal{A}(\triangle ABC)}, \quad w = \frac{\mathcal{A}(\triangle pAB)}{\mathcal{A}(\triangle ABC)}. \quad (2.16)$$

It directly follows that for any point  $p$  in the triangle, the barycentric coordinates are non-negative and that  $u + v + w = 1$ . Furthermore,  $u$  is constant on lines parallel to the edge  $BC$ , and attains its maximum value  $u = 1$  at  $p = A$ ; mutatis mutandis for  $v$  and  $w$ .

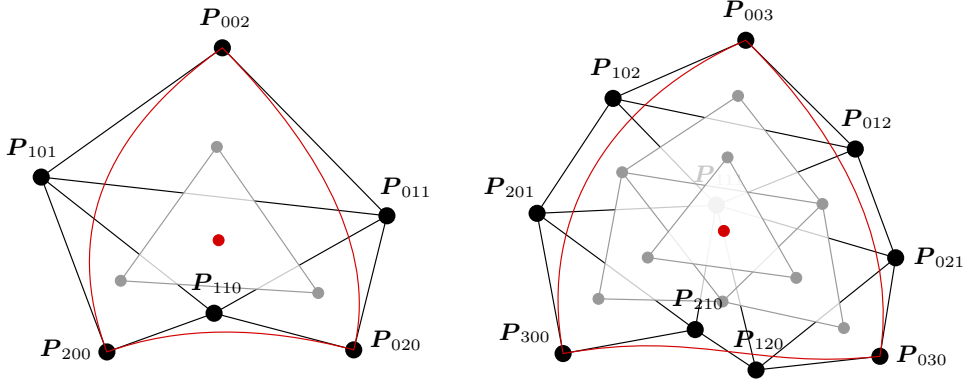
The construction of a triangular Bézier patch is now straightforward. Starting from a triangular control net — with a *type-I* triangulation<sup>‡</sup> — with control points  $\mathbf{P}_{klm}$  so that  $k, l, m \in [0, d]$  and  $k + l + m = d$ , we compute a fixed barycentric combination

$$u \mathbf{P}_{klm} + v \mathbf{P}_{k-1,l+1,m} + w \mathbf{P}_{k-1,l,m+1}$$

<sup>†</sup>Because of this definition, barycentric coordinates are also referred to as *area coordinates*. The definition can be directly generalised to higher-order simplices.

<sup>‡</sup>A type-I triangulation has triangles in two different orientations, which we refer to as 'up' ( $\triangle$ ) and 'down' ( $\nabla$ ).

for every triangle in the control net with its control points  $P_{klm}$ ,  $P_{k-1,l+1,m}$  and  $P_{k-1,l,m+1}$  ordered anti-clockwise. Like the construction for a quadrilateral patch, this results in a smaller grid, and repeating the procedure eventually results in a single point that lies on the surface patch. Figure 2.14 shows the construction.



**Figure 2.14:** Construction of a quadratic (left) and cubic triangular Bézier patch (right) using the triangular de Casteljau's algorithm. The evaluated point corresponds to  $(u, v, w) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$  in both cases.

In the quadratic case, we obtain the following –

$$\begin{aligned} T(u, v, w) &= u(uP_{200} + vP_{110} + wP_{101}) + \\ &\quad v(uP_{110} + vP_{020} + wP_{011}) + \\ &\quad w(uP_{101} + vP_{011} + wP_{002}) \\ &= u^2P_{200} + 2uvP_{110} + v^2P_{020} + 2uwP_{101} + 2vwP_{011} + w^2P_{002}. \end{aligned}$$

Just like in the univariate setting, we obtain the basis functions, in this case the *triangular Bernstein polynomials*  $\mathcal{B}_{klm}(u, v, w)$ , as a by-product. In general, they are defined as

$$\mathcal{B}_{klm}(u, v, w) = \binom{d}{k, l, m} u^k v^l w^m = \frac{d!}{k!l!m!} u^k v^l w^m. \quad (2.17)$$

Unlike the univariate setting, we do not need a superscript to indicate the degree of the Bernsteins, as  $d = k + l + m$ . The triangular Bézier patches can then be defined as

$$T(u, v, w) = \sum_{k+l+m=d} \mathcal{B}_{klm}(u, v, w) P_{klm}. \quad (2.18)$$

Regarding the *partial* derivatives of the triangular Bernsteins, we have

$$\frac{\partial \mathcal{B}_{klm}}{\partial u} = \frac{d!}{k!l!m!} k u^{k-1} v^l w^m = d \frac{(d-1)!}{(k-1)!l!m!} u^{k-1} v^l w^m = d \mathcal{B}_{k-1, l, m},$$

with similar expressions for partial derivatives with respect to  $v$  and  $w$ . Like before, Bernsteins with negative indices vanish. The *directional* derivative  $D_r \mathcal{B}_{klm}$ , with the direction  $r = (a_1, a_2, a_3)$  defined as the difference of two points expressed in barycentric coordinates, then follows as

$$D_r \mathcal{B}_{klm} = \nabla \mathcal{B}_{klm} \cdot r = d \left( a_1 \mathcal{B}_{k-1, l, m} + a_2 \mathcal{B}_{k, l-1, m} + a_3 \mathcal{B}_{k, l, m-1} \right). \quad (2.19)$$

The directional derivative of a triangular Bézier patch is obtained by substituting (2.19) into (2.18). For the quadratic case, this yields<sup>†</sup>

$$\begin{aligned} D_r T(u, v, w) &= d \left( a_1 u \mathbf{P}_{200} + (a_1 v + a_2 u) \mathbf{P}_{110} + a_2 v \mathbf{P}_{020} + \right. \\ &\quad \left. (a_1 w + a_3 u) \mathbf{P}_{101} + (a_2 w + a_3 v) \mathbf{P}_{011} + a_3 w \mathbf{P}_{002} \right) \\ &= d \left( u (a_1 \mathbf{P}_{200} + a_2 \mathbf{P}_{110} + a_3 \mathbf{P}_{101}) + \right. \\ &\quad \left. v (a_1 \mathbf{P}_{110} + a_2 \mathbf{P}_{020} + a_3 \mathbf{P}_{011}) + \right. \\ &\quad \left. w (a_1 \mathbf{P}_{101} + a_2 \mathbf{P}_{011} + a_3 \mathbf{P}_{002}) \right) \\ &= d \left( a_1 (u \mathbf{P}_{200} + v \mathbf{P}_{110} + w \mathbf{P}_{101}) + \right. \\ &\quad \left. a_2 (u \mathbf{P}_{110} + v \mathbf{P}_{020} + w \mathbf{P}_{011}) + \right. \\ &\quad \left. a_3 (u \mathbf{P}_{101} + v \mathbf{P}_{011} + w \mathbf{P}_{002}) \right). \end{aligned}$$

The above shows that the computation of the directional derivative of a triangular Bézier patch can be incorporated in the triangular de Casteljau algorithm, where either in the first or in the last step the barycentric combination using  $(u, v, w)$  is replaced by the barycentric combination using  $r = (a_1, a_2, a_3)$ . In fact, the same applies to the computation of the derivative in the univariate case.

Note that, by construction, the boundary curves of both the quadrilateral and triangular Bézier patch are Bézier curves themselves.

Finally, we remark that it is possible to construct Bézier patches on general  $n$ -gonal domains [LD89; VSK16] based on *generalised* barycentric coordinates (GBCs), though discussion of these constructions is postponed until Chapter 5.

### 2.2.2 B-spline patches

The generalisation of univariate B-splines (both uniform and non-uniform) to rectangular surface patches is straightforward by means of the tensor-product. Alternative definitions include simplex splines and box splines, which are discussed next.

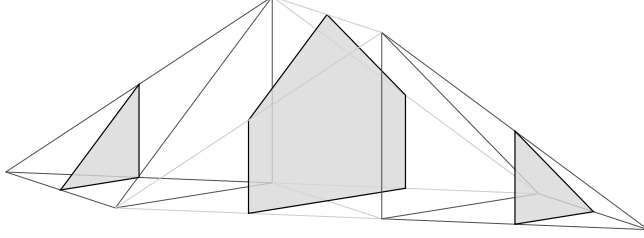
### 2.2.3 Simplex splines

Recall the construction of univariate uniform B-splines by means of convolution, discussed in Section 2.1.2. As mentioned, every value of  $t \in [0, d + 1]$  for a degree- $d$

<sup>†</sup>Note that  $\mathcal{B}_{100} = u$ ,  $\mathcal{B}_{010} = v$  and  $\mathcal{B}_{001} = w$ .



uniform B-spline is associated with the overlap of the unit pulse and the uniform B-spline of level  $d - 1$ . Taking  $d = 2$  as example, note that these overlaps, illustrated in Figure 2.9 on the right, can be stacked to form a polyhedron. The result is visualised in Figure 2.15.



**Figure 2.15:** Overlaps from the convolution process for constructing a uniform quadratic B-spline stacked to form a polyhedron.

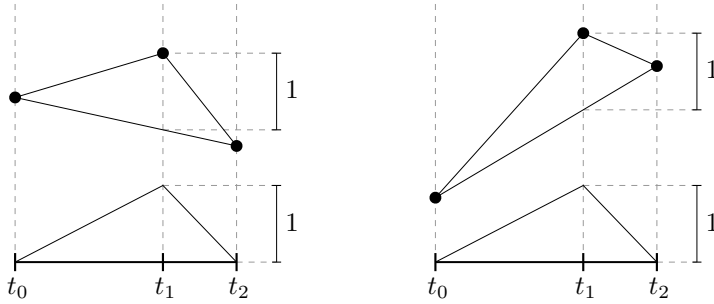
The key observation is that certain (piecewise) polynomial functions can evidently be defined as the cross-section of a polyhedron, or in other words, as its *projection*. The concept can be generalised to *polytopes* of any dimension and cross-sections of higher order (e.g. volumes for a polytope in  $\mathbb{R}^4$ ), resulting in *polyhedral splines*.

Arguably the most elementary subset of polyhedral splines are the *simplex splines*, i.e. projections of simplices [DM83]. Given a simplex of any dimension and a parametric domain of interest, we can consider cross-sections of the simplex orthogonal to this parametric domain in a continuous fashion. The result is a simplex spline, a piecewise polynomial function with knots corresponding to the positions where the cross-section intersects one or multiple vertices. Note that this definition includes projections not only to  $\mathbb{R}$ , but to any  $\mathbb{R}^n$ , the latter resulting in multivariate splines. Assuming non-degenerate simplices (e.g. excluding tetrahedra with all vertices co-planar), the degree of the resulting piecewise polynomial is  $d = m - n$ , with  $m$  the dimension of the simplex and  $n$  the dimension of the parameter space projected onto. The parametric continuity between segments is  $C^{d-v}$ , with  $v$  the number of vertices in the cross-section associated with the connection.

The above definition implies that univariate B-splines (i.e. both uniform and non-uniform ones) are a subset of simplex splines. Indeed, taking  $d = 1$  as example, linear B-splines can be defined as projections of triangles. This means that linear B-splines can be defined using 3 knots. To see this, consider a sequence of knots  $[t_0, t_1, t_2]$  on a univariate parameter domain  $\subset \mathbb{R}$ . Next, we place the three vertices defining a triangle on lines orthogonally intersecting the parameter domain at our knots. The length of the intersection of our triangle and lines orthogonal to the parameter domain results in our function values. Now, observe that that we cannot choose just *any* triangle. After all, a set of linear B-splines should form a partition of unity, which — in this case — means that the function value at the middle knot of each linear B-spline should be 1. In other words, the length of the intersection of our triangle and the line perpendicular to the parameter domain at  $t_1$  should be 1. This splits the triangle into two sub-triangles, each with a base  $b$  of length 1 and heights  $h_L, h_R$  of  $t_1 - t_0$  and  $t_2 - t_1$ , respectively. It follows that only triangles with an area of  $A = \frac{1}{2}b(h_L + h_R) = \frac{t_2 - t_0}{2}$  project to linear

B-splines. Figure 2.16 illustrates the construction.

A similar approach can be taken for  $d = 2$ , showing that quadratic B-splines can be defined as projections of tetrahedra. We now consider a sequence of 4 knots  $[t_0, t_1, t_2, t_3]$  on a univariate parameter domain. The vertices defining a tetrahedron should be placed on planes orthogonally intersecting our parameter domain at the knots. The values of the simplex spline now follow as areas of intersections of our tetrahedron and planes orthogonal to the parameter domain. In this case, determining the conditions so that a set of quadratic B-splines form a partition of unity is considerably less trivial. Ultimately, it turns out that only tetrahedra with a volume of  $V = \frac{1}{2}Ah = \frac{t_3 - t_0}{3}$  project to quadratic B-splines. For general degree  $d$  B-splines, the condition is that the (hyper-)volume of the projected simplex should equal  $\frac{t_{d+1} - t_0}{d+1}$  [PBP02].



**Figure 2.16:** Infinitely many triangles project to the same linear B-spline associated with knots  $[t_0, t_1, t_2]$ .

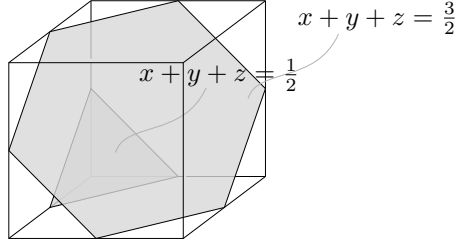
Another subset of simplex splines are the *normalised* simplex splines — the area of the cross-sections is divided by the volume of the simplex, resulting in simplex splines with a unit integral. This makes them scale-invariant. Uniform B-splines are in this subset as they have a unit integral<sup>†</sup>.

Somewhat surprisingly, simplex splines no longer appear to be an area of much active research, even though the topic seems far from exhausted.

### 2.2.4 Box splines

The next subset of polyhedral splines we consider are the *box splines* [BHR93; PB02]. As the name suggests, these splines are projections of boxes (hypercubes) in  $\mathbb{R}^m$  onto  $\mathbb{R}^n$ . They can be regarded as generalisations of (uniform) B-splines. However, instead of a knot-vector, box splines are defined using a *direction matrix*  $\Xi$ . The columns of this matrix are the direction vectors  $\xi_k \in \mathbb{R}^n$ , with  $k \in [1, m] \subset \mathbb{Z}$ , which can be interpreted as parameter intervals. The direction matrix also specifies the orientation of the cross-sections. For example, the uniform quadratic B-spline  $\mathcal{M}^2(t)$  is associated with  $\Xi = (1 \ 1 \ 1)$ . It is therefore the projection of a cube, and the cross-sections are oriented such that  $p \in \mathbb{R}^{m=3} : \Xi p = t$ . Figure 2.17 illustrates the example.

<sup>†</sup>The integral of a function  $h$  constructed through the convolution of functions  $f$  and  $g$  equals the product of the integrals of  $f$  and  $g$ . As any uniform B-spline is the result of repeated convolution of the unit pulse with itself, it follows that its integral is 1.



**Figure 2.17:** The uniform quadratic B-spline defined as a box spline. The orientation of the cross sections of the (unit) cube are determined by the direction matrix and are in this case orthogonal to the main diagonal of the cube.

The *support* of a box spline  $\mathcal{M}_\Xi$ , i.e. the region in  $\mathbb{R}^n$  where it is nonzero, can be defined as the Minkowski sum of its direction vectors,

$$\text{supp}(\mathcal{M}_\Xi) = \sum_{k=1}^m \lambda_k \xi_k : 0 \leq \lambda_k \leq 1, \quad (2.20)$$

which is a *zonotope*. Although this means that arbitrary direction vectors can be used, in practice they are selected such that the gridlines in the resulting support agree with a regular tessellation. In the case of  $n = 2$ , this results in *two-directional* box splines whose supports admit square grids, and *three-directional* and *four-directional* box splines, whose supports are type-I and type-II triangulations, respectively<sup>†</sup>. Short-hand notation can be introduced for these box splines, where instead of showing  $\Xi$  as the subscript, a vector of length 2, 3 or 4 containing the multiplicities of the direction vectors is used. For instance, the two-directional box spline corresponding to  $\Xi = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$ , also known as the uniform biquadratic B-spline, can be written as  $\mathcal{M}_{33}$ . Likewise, the three-directional box spline corresponding to  $\Xi = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ , can be written as  $\mathcal{M}_{111}$ .

An alternative method to define box splines is by generalising the approach based on convolution used to define uniform B-splines. However, in this case we apply *directional convolution*, that is, convolution with the unit pulse aligned with a direction vector  $\xi_k$ . It follows that (2.11) generalises to

$$\mathcal{M}_\Xi(u) = \int_0^1 \mathcal{M}_{\Xi \setminus \xi_k}(u - \tau \xi_k) d\tau, \quad (2.21)$$

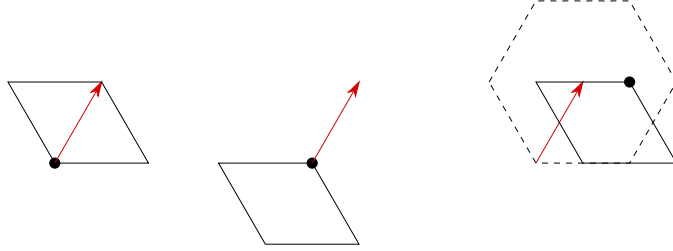
where it should be noted that both  $u \in \mathbb{R}^n$  and  $\xi_k \in \mathbb{R}^n$ , but  $\tau \in \mathbb{R}$ . Similarly, the derivative of a uniform B-spline (2.12) generalises to the *directional derivative*  $D_{\xi_k}$  of a box spline,

$$D_{\xi_k} \mathcal{M}_\Xi(u) = \mathcal{M}_{\Xi \setminus \xi_k}(u) - \mathcal{M}_{\Xi \setminus \xi_k}(u - \xi_k). \quad (2.22)$$

Considering bivariate box splines, the initial function (i.e. the equivalent of the unit pulse for uniform B-splines) is the indicator function  $\mathcal{M}_{11}$ . The directions associated

<sup>†</sup>Three- or four-directional box splines should not be confused with *triangular* B-splines [DMS92] (also referred to as DMS splines), which are a different generalisation of B-splines not discussed here.

with two-directional box splines, commonly labelled as  $e_1$  and  $e_2$ , are usually chosen as two orthogonal unit vectors, though for three-directional box splines, they are often *visualised* as unit vectors that are  $120^\circ$  apart; this highlights the symmetry of a type-I triangulation. Figure 2.18 shows the convolution of  $\mathcal{M}_{11}$  with  $\xi_k = e_3 = e_1 + e_2$ , which results in  $\mathcal{M}_{111}$ .



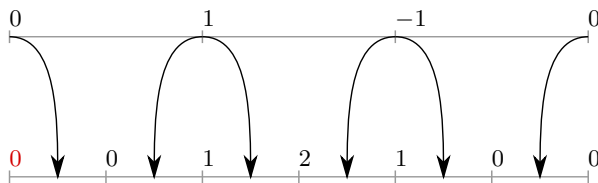
**Figure 2.18:** Directional convolution of  $\mathcal{M}_{11}$  (rhombus) with  $e_3$  (red arrow), resulting in  $\mathcal{M}_{111}$  (its support is outlined using dashed edges).

Box splines can be defined and evaluated using either method<sup>†</sup>, but this might not always be the most practical approach to evaluate them. Focusing again on bivariate box splines, recall that for the two-, three- or four-directional ones, the gridlines in the support of such a box spline partition it into squares or triangles. Every such square or triangle corresponds to a polynomial (not *piecewise* polynomial) part of the box spline. These parts can therefore be expressed as Bézier patches, which results in the *Bernstein-Bézier* (BB) representation of a box spline.

Unfortunately, the concept of blossoming does not seem to extend to general box splines (although it does generalise to triangular Bézier patches), which would have provided a means to obtain the BB-representation. We temporarily return to the univariate setting, where in addition to blossoming, there turns out to be an alternative method to obtain the BB-coefficients of a B-spline. Taking the uniform quadratic B-spline  $\mathcal{M}^2(t)$  as example again, we can apply (2.12) to obtain its derivative expressed as the difference of two uniform linear B-splines. The BB-coefficients  $d_l$  of the latter are trivial. Then, by expressing each segment of  $\mathcal{M}^2(t)$  as a Bézier curve with unknown BB-coefficients  $c_l$ , using (2.8) we can connect these to the BB-coefficients of  $\mathcal{M}^1(t) - \mathcal{M}^1(t - 1)$ . This results in  $d_l = 2(c_{l+1} - c_l)$  for the first segment,  $d_l = 2(c_{l+2} - c_{l+1})$  for the second segment, and  $d_l = 2(c_{l+3} - c_{l+2})$  for the third. This gives us  $2 \cdot 3 = 6$  equations for 7 unknowns  $c_l$ . However, we also know that  $c_0 = 0$  for any uniform B-spline. This is enough information to compute the BB-coefficients; Figure 2.19 illustrates the approach. Once the BB-coefficients of  $\mathcal{M}^2(t)$  are known, they can be used to compute those of  $\mathcal{M}^3(t)$ , and in general to obtain the BB-coefficients of any  $\mathcal{M}^d(t)$  by applying the procedure iteratively.

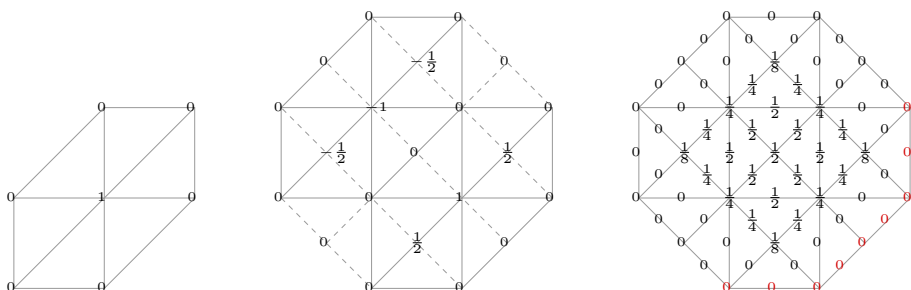
The procedure generalises to box splines, now using (2.22) for the (directional) derivative, and applying (2.19) to connect the two sets of BB-coefficients. As example we consider the piecewise quadratic four-directional box spline  $\mathcal{M}_{1111}(u)$ , also known as the *Zwart-Powell* spline [Zwa73]. Taking its partial derivative in the direction of

<sup>†</sup>In addition, there are several alternative definitions of box splines, see [BHR93].



**Figure 2.19:** The derivative of  $\mathcal{M}^2(t)$  expressed as the difference of two uniform linear B-splines with BB-coefficients  $d_0 = 0, d_1 = 1, d_2 = -1$  and  $d_3 = 0$  (top). These coefficients can be used to obtain the BB-coefficients  $c_l$  for  $2\mathcal{M}^2(t)$  (bottom), when we also apply  $c_0 = 0$ .

$e_4 = -e_1 + e_2$  results in the difference  $\mathcal{M}_{111}(u) - \mathcal{M}_{111}(u - e_4)$ . As  $\mathcal{M}_{111}$  is piecewise linear, its BB-coefficients  $d_{kl}$  are trivial. Expressing all triangular parts of  $\mathcal{M}_{111}$  in Bernstein-Bezier form with unknown coefficients  $c_{kl}$ , we see that for an upward-pointing type-II triangle,  $e_4$  corresponds to  $(0, -2, 2)$  in barycentric coordinates. This means that differences of neighbouring coefficients  $c_{kl}$  on gridlines in this direction match the coefficients  $(d_{kl})/4$ . The observation also holds for the type-II triangles in the other three orientations. The procedure is shown in Figure 2.20. Using it iteratively allows the computation of the BB-coefficients for all three- and four-directional box splines.



**Figure 2.20:** Obtaining the BB-coefficients of the Zwart-Powell spline using the coefficients of its directional derivative in the direction of  $e_4$ .

From the BB-coefficients of a three- or four directional box spline it is readily observed that its triangular parts (i.e. the three or four sets of triangles oriented in the same direction) partition unity when they are overlapped. As such, they can be used as *blending functions* to represent triangular surface patches. Alternatively, instead of overlapping merely the individual parts, we can place copies of the entire box spline on a (three- or four-directional) grid such that the centre of support of each copy matches a gridline intersection. This way, we achieve the same overlap of individual parts, with the important difference that each individual part now comes from a different box spline copy. Not only does this reveal the natural pattern of the control points associated with the blending functions, it also results in a *box spline surface* that is composed of triangular patches (meeting with some order of parametric continuity) upon associating every copy with a control point. The next section, discussing *half-box splines*, contains an illustrated example of this concept.

Note that the above approach also applies to both univariate and bivariate uniform B-splines (see e.g. Figure 2.7). Tensor-product uniform B-splines are a subset of two-directional box splines — bivariate non-uniform B-splines or NURBS, however, are not.

### 2.2.5 Half-box splines

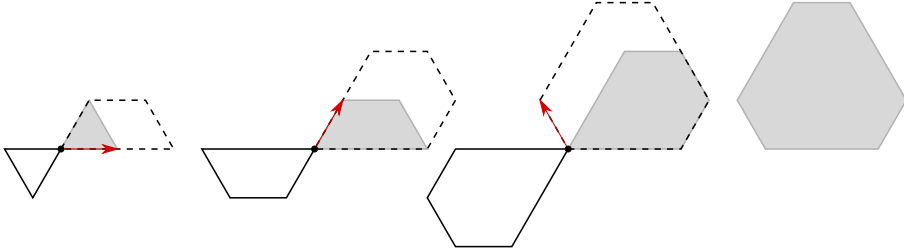
The last subset of polyhedral splines we look into are the half-box splines [Fre71; Sab77; Pra84]. These are indeed very similar to box splines — using the projection-based definition, only *half* of a hypercube is projected in this case. Regarding the convolution-based definition, the initial function is now a triangle-shaped indicator function — in other words, only *half* of the rhombus-shaped  $\mathcal{M}_{11}$  that is the starting point for defining bivariate box splines through directional convolution.

Considering both halves individually using either definition results in two piecewise polynomial functions that are mirror-symmetric. Clearly, their sum equals the box spline associated with the entire hypercube (projection) or rhombus (convolution).

In a bivariate setting, it makes sense to consider three- and four-directional half-box splines; we focus on the former. In that case, the initial setting is an indicator function with an up-oriented equilateral triangle  $\triangle$  as support, which can then be convolved in any of the three directions  $e_1, e_2$  or  $e_3$ :

$$H_{\Xi}(u) = \int_0^1 H_{\Xi \setminus e_k}(u - \tau e_k) d\tau. \quad (2.23)$$

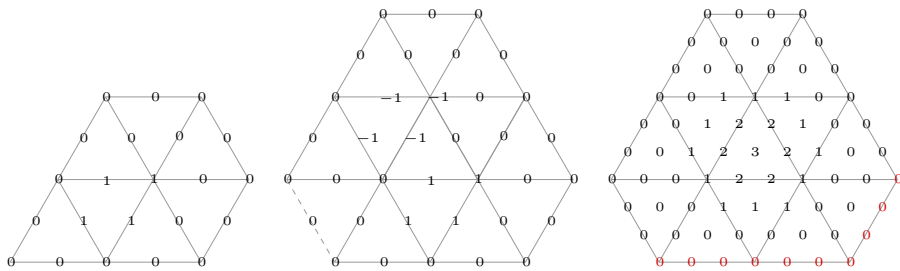
We choose to convolve once in each of the three directions, and obtain a half-box spline for which we use the shorthand notation  $H_{111}(u)$ . It is a piecewise cubic function for which the parts connect with  $C^1$  continuity; see Figure 2.21.



**Figure 2.21:** Using directional convolution to construct the half-box spline  $H_{111}(u)$ . In each of the three steps, the support of  $H_{\Xi \setminus e_k}$  (grey) is rotated by  $180^\circ$  (i.e. the generalisation of reflecting a function in the case of univariate convolution). This rotated version (black) is then translated by  $u \in \mathbb{R}^2$ , after which its overlap with the current direction vector  $e_k$  (red) is integrated. The support of the resulting function  $H_{\Xi}(u)$  is outlined (dashed).

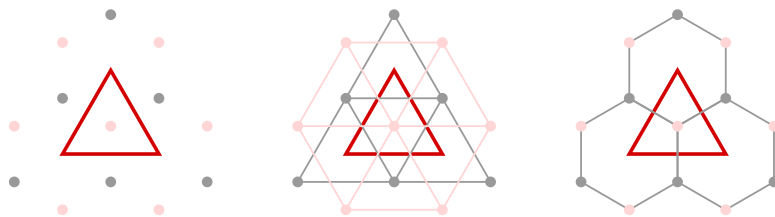
Starting instead from the down-oriented equilateral triangle  $\nabla$  and using the same construction, we obtain the mirror-symmetric half-box spline  $\bar{H}_{111}(u)$ . It follows that  $H_{111}(u) + \bar{H}_{111}(u) = \mathcal{M}_{221}(u)$ .

The procedure followed to obtain the BB-coefficients of box splines can also be applied to half-box splines [Far82b; Böh83]. The last step in this iterative process to compute the BB-coefficients of  $H_{111}(u)$  is illustrated in Figure 2.22.



**Figure 2.22:** The BB-coefficients of  $2H_{110}(u)$  (left) and the directional derivative of  $6H_{111}(u)$  in the direction of  $e_3$  expressed as the difference of two shifted copies of  $2H_{110}(u)$  (middle), see (2.22). Setting the BB-coefficients on the bottom-right boundary of  $6H_{111}$  to zero (red), its other coefficients can be determined from those of its directional derivative (right).

From the Bernstein-Bézier representation of  $H_{111}(u)$ , we observe that the 13 up-oriented triangles of  $H_{111}(u)$  and  $\bar{H}_{111}(u)$  partition unity. Naturally, the same holds for the 13 down-oriented triangles. In fact, because of this symmetry, we can simply overlap all triangles of only  $H_{111}(u)$  to arrive at the same result. Overlapping (rotated) copies of  $H_{111}(u)$  on the three-directional grid, while keeping track of their centres<sup>†</sup>, then reveals the natural pattern of the control points associated with a half-box spline surface patch, see Figure 2.23 (left). This example nicely shows that connections between the control points (to form the control net) are artificial. In this particular case, we have two ways to connect the points — either based on the original orientation of each triangular part, resulting in two separate type-I triangular control nets, or a single *hexagonal* control net; see Figure 2.23.

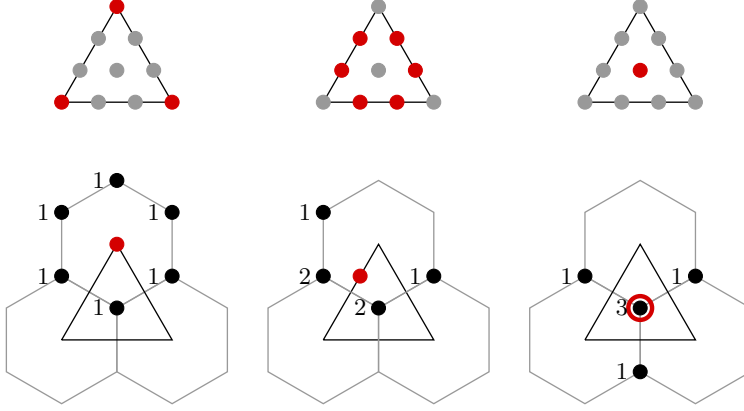


**Figure 2.23:** The natural pattern of control points associated with a triangular half-box spline surface patch (left). They can be connected to form either two separate triangular control nets (middle) or a single hexagonal net (right).

Using the single hexagonal control net seems to be the favourable choice, as it is considerably less cluttered compared to using the two separate triangular ones. At the same time, it is a curiosity in the bivariate spline world dominated by quadrilaterals and triangles. Moreover, it shows aspects of a *dual* nature, as the shape of the surface patch does not match that of (the components of) its control net, but instead is its dual. Nevertheless, the half-box spline patch can easily be represented as a triangular Bézier

<sup>†</sup>These centres are commonly referred to as *Greville abscissae*.

patch — the corresponding control points can be computed using the *stencils* (i.e. affine combinations) shown in Figure 2.24.



**Figure 2.24:** The Bézier stencils for the BB-representation of a cubic half-box spline patch directly follow from the BB-coefficients in Figure 2.22. Normalisation of the coefficients by a factor of 6 is implied.

Based on the above, I conjecture the existence of a spline patch associated with a control net composed of octagons and squares (in other words, a control net that is dual to a type-II triangulation). The corresponding blending functions might be four-directional half-box splines, though perhaps other *fractional*-box splines have to be considered.

### 2.2.6 Alternative representations

As we have seen, Bézier patches form the elementary building blocks for representing several other splines and spline patches. However, various other directions to construct bivariate surface patches have been explored. In addition to Lagrange patches, we consider Coons' approach, which — when applied to cubic polynomials — results in Ferguson patches and Hermite patches as special cases. Detailed discussion of other patches, including Gregory patches, is postponed until Chapter 3.

#### Lagrange patches

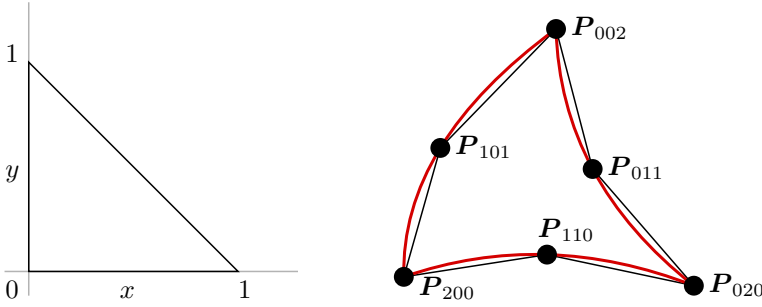
The principle of Lagrange curves can be extended to both quadrilateral and triangular patches. In the former case, we can construct the basis functions for a degree  $d$  patch by means of the tensor-product. Considering the unit square  $(u, v) \in [0, 1]^2$  as the parameter domain, we can select a set of  $d + 1$  values  $u_k$  and  $d + 1$  values  $v_k$  that control where in parameter space our control points  $\mathbf{P}_{kl}$  are interpolated. The basis functions are then defined by the tensor product of (2.13) with itself.

Taking  $d = 2$  as example, the Lagrange polynomial associated with the control point  $\mathbf{P}_{21}$  is  $\mathcal{L}_{2,1}^2(u, v) = \frac{u-u_0}{u_2-u_0} \frac{u-u_1}{u_2-u_1} \frac{v-v_0}{v_1-v_0} \frac{v-v_2}{v_1-v_2}$ . Assuming equidistant values  $u_k$  and  $v_k$ , this becomes  $\mathcal{L}_{2,1}^2(u, v) = -4u(2u-1)v(v-1)$ .



The basis functions for a triangular Lagrange patch can also be constructed using the Kronecker delta property that is so characteristic for the Lagrange bases. With the *unit triangle* as parameter domain (see Figure 2.25), we can obtain the basis functions  $\mathcal{L}_{klm}(x, y)$  by multiplying the expressions for the lines where they should vanish, and subsequently scaling the result such that their value equals 1 at the remaining point. For example, the quadratic  $\mathcal{L}_{110}(x, y)$  should vanish on the line  $x = 0$  and on the line  $x = 1 - y$  or  $1 - x - y = 0$ . It should then be scaled by a factor of 4. The expressions for all quadratic basis functions are

$$\begin{aligned}\mathcal{L}_{200}(x, y) &= (1 - x - y)(1 - 2x - 2y), & \mathcal{L}_{110}(x, y) &= 4x(1 - x - y), \\ \mathcal{L}_{020}(x, y) &= x(2x - 1), & \mathcal{L}_{011}(x, y) &= 4xy, \\ \mathcal{L}_{002}(x, y) &= y(2y - 1), & \mathcal{L}_{101}(x, y) &= 4y(1 - x - y).\end{aligned}$$



**Figure 2.25:** The unit triangle (left) and a quadratic Lagrange patch with its control net (right).

Note that for this particular triangle, the barycentric coordinates can be expressed as  $u = 1 - x - y$ ,  $v = x$  and  $w = y$ . Rewriting the expressions for the basis given above then reveals its (barycentric) symmetry:

$$\begin{aligned}\mathcal{L}_{200}(u, v, w) &= u(2u - 1), & \mathcal{L}_{110}(u, v, w) &= 4uv, \\ \mathcal{L}_{020}(u, v, w) &= v(2v - 1), & \mathcal{L}_{011}(u, v, w) &= 4vw, \\ \mathcal{L}_{002}(u, v, w) &= w(2w - 1), & \mathcal{L}_{101}(u, v, w) &= 4uw.\end{aligned}$$

Similar to Bézier patches, the boundary curves of both quadrilateral and triangular Lagrange patches are Lagrange curves.

### Coons patches

The aim of Coons patches is to interpolate a given set of boundary curves meeting at common corners  $P_{kl}$  resulting in a surface patch parameterised on the unit square. The concept is sometimes referred to as *transfinite interpolation*.

For a quadrilateral patch, we have two given curves in the  $u$ -direction,  $A(u)$  and  $B(u)$ , and likewise two given curves in the  $v$ -direction,  $C(v)$  and  $D(v)$ . These curves

are connected such that

$$\begin{aligned} A(0) &= \mathbf{P}_{00}, & B(0) &= \mathbf{P}_{01}, & C(0) &= \mathbf{P}_{00}, & D(0) &= \mathbf{P}_{10}, \\ A(1) &= \mathbf{P}_{10}, & B(1) &= \mathbf{P}_{11}, & C(1) &= \mathbf{P}_{01}, & D(1) &= \mathbf{P}_{11}. \end{aligned}$$

Next, we construct two *ruled surfaces* (also known as *lofted surfaces*)

$$\begin{aligned} R_1(u, v) &= (1 - v)A(u) + vB(u), \\ R_2(u, v) &= (1 - u)C(v) + uD(v). \end{aligned}$$

Simply adding  $R_1(u, v)$  and  $R_2(u, v)$  does not result in the interpolation of the given boundary curves<sup>†</sup>, but if we subtract the bilinear surface

$$L(u, v) = (1 - v)((1 - u)\mathbf{P}_{00} + u\mathbf{P}_{10}) + v((1 - u)\mathbf{P}_{01} + u\mathbf{P}_{11}),$$

the boundary is indeed interpolated. As such, we end up with a *bilinearly blended* Coons patch,

$$S(u, v) = R_1(u, v) + R_2(u, v) - L(u, v). \quad (2.24)$$

In a similar fashion, we can construct a *triangular* Coons patch. Using barycentric coordinates, we assume that the three given boundary curves are defined as  $A(v, w)$ ,  $B(u, w)$  and  $C(u, v)$  such that

$$\begin{aligned} A(1, 0) &= \mathbf{P}_{010}, & B(1, 0) &= \mathbf{P}_{001}, & C(1, 0) &= \mathbf{P}_{100}, \\ A(0, 1) &= \mathbf{P}_{001}, & B(0, 1) &= \mathbf{P}_{100}, & C(0, 1) &= \mathbf{P}_{010}. \end{aligned}$$

Next, we construct three surfaces

$$\begin{aligned} T_A(u, v, w) &= u\mathbf{P}_{100} + (1 - u)A(v, w), \\ T_B(u, v, w) &= v\mathbf{P}_{010} + (1 - v)B(u, w), \\ T_C(u, v, w) &= w\mathbf{P}_{001} + (1 - w)C(u, v). \end{aligned}$$

With  $L(u, v, w) = u\mathbf{P}_{100} + v\mathbf{P}_{010} + w\mathbf{P}_{001}$ , we can then define a linearly blended triangular Coons patch as

$$T(u, v, w) = T_A(u, v, w) + T_B(u, v, w) + T_C(u, v, w) - 2L(u, v, w). \quad (2.25)$$

We note that there are several alternative constructions for triangular Coons patches<sup>‡</sup>, including one based on interpolation between pairs of boundary curves.

<sup>†</sup>Note that the boundary curves are not required to be polynomial, but can instead be arbitrary parametric curves. This is a strong point of Coons patches.

<sup>‡</sup>An obvious alternative would be to simply merge two points of a quadrilateral patch. However, this is not the cleanest approach, as artefacts in position or derivative might occur.

## Ferguson's patches

Instead of using linear interpolation between the provided boundary curves, we can generalise (2.24) and interpolate the boundaries using another pair of blending functions matching the following requirements—they should partition unity, and moreover, actually interpolate the corners. A popular choice is to use the cubic Hermite basis functions  $\mathcal{H}_0^3(t)$  and  $\mathcal{H}_3^3(t)$  (see Section 2.1.3), which satisfy  $\mathcal{H}_0^3(t) + \mathcal{H}_3^3(t) = 1 \forall t \in [0, 1]$ , as well as the second requirement. The concept remains the same, and we obtain

$$S(u, v) = \mathcal{H}_0^3(v)A(u) + \mathcal{H}_3^3(v)B(u) + \mathcal{H}_0^3(u)C(v) + \mathcal{H}_3^3(u)D(v) - \left( \mathcal{H}_0^3(v)(\mathcal{H}_0^3(u)\mathbf{P}_{00} + \mathcal{H}_3^3(u)\mathbf{P}_{10}) + \mathcal{H}_3^3(v)(\mathcal{H}_0^3(u)\mathbf{P}_{01} + \mathcal{H}_3^3(u)\mathbf{P}_{11}) \right),$$

which is the *partially bicubically blended* Coons patch.

Something interesting happens when the boundary curves are cubic Hermite curves; recall that these curves require two points  $\mathbf{P}$  as well as two *tangent vectors*  $\mathbf{T}$  as input. Writing the Coons patch in matrix form then gives us

$$S(u, v) = \begin{pmatrix} \mathcal{H}_0^3(u) & \mathcal{H}_1^3(u) & \mathcal{H}_2^3(u) & \mathcal{H}_3^3(u) \end{pmatrix} \begin{pmatrix} \mathbf{P}_{00} & \mathbf{T}_{00}^v & \mathbf{T}_{01}^v & \mathbf{P}_{01} \\ \mathbf{T}_{00}^u & 0 & 0 & \mathbf{T}_{01}^u \\ \mathbf{T}_{10}^u & 0 & 0 & \mathbf{T}_{11}^u \\ \mathbf{P}_{10} & \mathbf{T}_{10}^v & \mathbf{T}_{11}^v & \mathbf{P}_{11} \end{pmatrix} \begin{pmatrix} \mathcal{H}_0^3(v) \\ \mathcal{H}_1^3(v) \\ \mathcal{H}_2^3(v) \\ \mathcal{H}_3^3(v) \end{pmatrix}, \quad (2.26)$$

with  $\mathbf{T}_{kl}^u$  the tangent vector at point  $\mathbf{P}_{kl}$  for a boundary curve in the  $u$ -direction and  $\mathbf{T}_{kl}^v$  defined similarly. The resulting patch is known as a *Ferguson patch*.

## Hermite patches

With Hermite curves in mind, another generalisation of Coons patches can be derived by studying (2.26). Instead of simply interpolating between two boundary curves  $A(u)$  and  $B(u)$ , we could add two tangent functions  $A_T(u)$  and  $B_T(u)$  that specify the tangent vector in the  $v$ -direction along  $A(u)$  and  $B(u)$ , and simultaneously blend those using two blending functions that partition *nullity*<sup>†</sup>. The remaining two cubic Hermite basis functions are a suitable choice for this purpose as  $\mathcal{H}_1^3(t) + \mathcal{H}_2^3(t) = 0 \forall t \in [0, 1]$ . We note that  $A_T(u)$  and  $B_T(u)$  are commonly referred to as *tangent ribbons*, as they define strips or *ribbons* of tangents along the curves.

We apply the approach in both parametric directions. Now, in order to satisfy all input data, we have to subtract a patch constructed using this data and the blending functions we used (cf. the construction of the linearly blended Coons patch). It follows that this subtracted patch is the previously defined Ferguson patch; the resulting patch is called a *(fully) bicubically blended* Coons patch.

We remark that in the above definition, the boundary curves and tangent ribbons can be arbitrary parametric curves. If the boundary curves are again cubic Hermite curves, and the tangent ribbons are defined using only the Hermite tangent vectors

<sup>†</sup>This is not the only requirement — the partial derivatives of the resulting patch should of course interpolate the provided tangents. The cubic Hermite functions satisfy this condition.

from the other directions (i.e.  $A_T(u) = \mathcal{H}_0^3(u)\mathbf{T}_{00}^v + \mathcal{H}_3^3(u)\mathbf{T}_{10}^v$ ), we end up with a Ferguson patch again.

An alternative is to define  $A_T(u)$  and the other tangent ribbons as

$$A_T(u) = \mathcal{H}_0^3(u)\mathbf{T}_{00}^v + \mathcal{H}_1^3(u)\mathbf{D}_{00}^{vv} + \mathcal{H}_2^3(u)\mathbf{D}_{10}^{vv} + \mathcal{H}_3^3(u)\mathbf{T}_{10}^v,$$

where  $\mathbf{D}_{kl}^{vv}$  can be interpreted as second-order derivative information. As everything is now expressed in Hermite form, the resulting patch is a tensor-product Hermite patch (i.e. a bicubic Hermite patch). Its matrix form is similar to (2.26), but with the grey zeroes replaced by  $\mathbf{D}_{kl}^{uu} + \mathbf{D}_{kl}^{vv}$ . These entries are the mixed second-order partial derivatives of the patch, and are known as *twist vectors*. From this perspective, the Ferguson patches we discussed are Hermite patches with vanishing twists.

To conclude, we mention that also triangular Coons patches can be extended to cubically blended ones, though their construction is not discussed here.

## 2.3 Refinement

In the process of designing a curve or surface, it often happens that the user requires more control points to (locally) change the shape of an object. This also implies that the set of associated basis functions changes—not only do we get more basis functions, but some of them have to be changed such that the basis still partitions unity. Modification of the basis functions, or rather their refinement, is also an important aspect of numerical methods (as we will see in Chapter 4). In this section, we discuss a couple of techniques for the refinement of both control structure and basis.

### 2.3.1 Knot insertion

Our first look at refinement considers the notion of *knot insertion*, which can be interpreted in essentially two different ways; both are explored below.

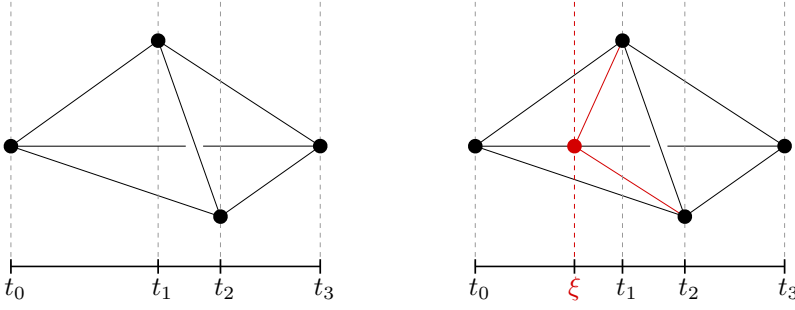
#### Minimal refinement

Recall that a (possibly non-uniform) B-spline can be defined as the projection of a simplex (see Section 2.2.3). In the following, we focus — without losing generality — on univariate quadratic B-splines associated with knots  $[t_0, t_1, t_2, t_3]$ .

Observe that the tetrahedron  $\mathcal{T}$  with volume  $V = \frac{t_3 - t_0}{3}$  defining a quadratic B-spline  $\mathcal{M}(t)$  can be split into two tetrahedra  $\mathcal{T}_a$  and  $\mathcal{T}_b$  by introducing a new vertex on the edge connecting the two vertices projecting to the first and last knot of  $\mathcal{M}(t)$ , respectively. The split is completed by connecting the new vertex to the other two vertices. Note that if the new vertex was introduced on any other edge, one of the resulting tetrahedra would be degenerate. See Figure 2.26.

The key observation is that  $\mathcal{T}_a$  and  $\mathcal{T}_b$  project to piecewise quadratic functions  $\mathcal{M}_a(t)$  and  $\mathcal{M}_b(t)$ , respectively, such that  $\mathcal{M}_a(t) + \mathcal{M}_b(t) = \mathcal{M}(t)$ . This is a powerful result that holds in general — *any simplex spline can be regarded as the sum of two other simplex splines*. I refer to this concept as *minimal refinement*.

Note that the volumes of  $\mathcal{M}_a(t)$  and  $\mathcal{M}_b(t)$  are  $V_a = \frac{\xi - t_0}{t_3 - t_0} V$  and  $V_b = \frac{t_3 - \xi}{t_3 - t_0} V$ , respectively (the ratios follow from the subdivision of the base of  $\mathcal{T}$ , the height of both



**Figure 2.26:** Splitting a simplex into two smaller simplices by introducing a new vertex (red).

sub-tetrahedra is the same as that of  $\mathcal{T}$ ). Regarding the position of  $\xi$ , we have the following cases<sup>†</sup>:

$$\begin{aligned} t_0 < \xi < t_1 : \quad V_a &= \frac{\xi - t_0}{3} < \frac{t_2 - t_0}{3}, & V_b &= \frac{t_3 - \xi}{3}, \\ t_1 < \xi < t_2 : \quad V_a &= \frac{\xi - t_0}{3} < \frac{t_2 - t_0}{3}, & V_b &= \frac{t_3 - \xi}{3} < \frac{t_3 - t_1}{3}, \\ t_2 < \xi < t_3 : \quad V_a &= \frac{\xi - t_0}{3}, & V_b &= \frac{t_3 - \xi}{3} < \frac{t_3 - t_1}{3}. \end{aligned}$$

This shows that in most cases,  $\mathcal{M}_a(t)$  and  $\mathcal{M}_b(t)$  are *scaled* B-splines, an important detail which we will touch upon below.

### Regular refinement

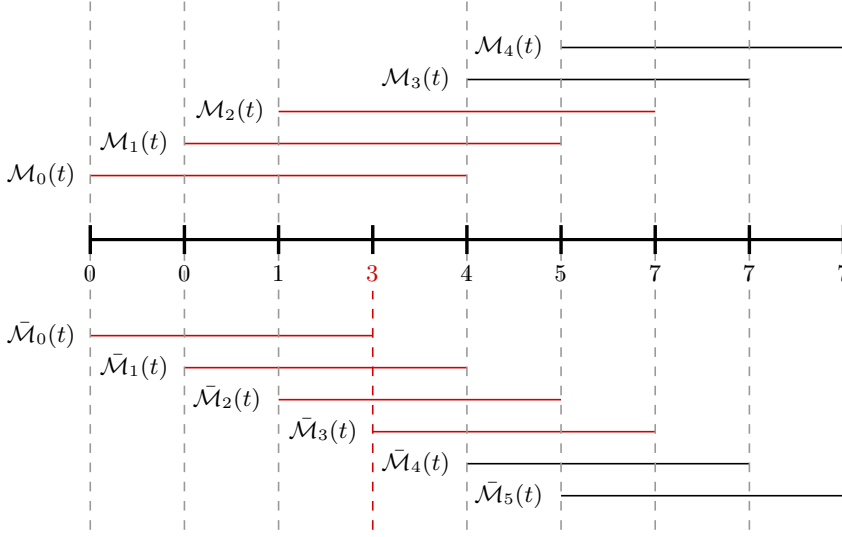
Consider now a knot-vector  $\Xi = [0, 0, 1, 4, 5, 7, 7, 7]$ , which for  $d = 2$  defines 5 non-uniform quadratic B-splines  $\mathcal{M}_k(t)$ . Each B-spline is then associated with a control point  $P_k$ , such that a curve  $C(t) = \sum_{k=0}^4 \mathcal{M}_k(t) P_k$  can be defined.

We decide to insert the knot  $\xi = 3$  into  $\Xi$ , which then defines 6 quadratic B-splines  $\bar{\mathcal{M}}_k(t)$ . How are these basis functions defined? And where does the extra control point come from (after all, we have one more basis function, and each of those should be associated with a control point)?

The first question is straightforward to answer, as we already saw that each quadratic B-spline is defined using 4 consecutive knots (also called *local knot vectors*) from  $\Xi$ , sharing  $d + 1 = 3$  knots with the next B-spline. Looking at the new knot-vector  $\bar{\Xi} = [0, 0, 1, 3, 4, 5, 7, 7, 7]$ , we see that the new knot appears in the 4 local knot vectors defining the new  $\bar{\mathcal{M}}_0(t), \dots, \bar{\mathcal{M}}_3(t)$ . The remaining local knot vectors are not affected, so  $\bar{\mathcal{M}}_4(t) = \mathcal{M}_3(t)$  and  $\bar{\mathcal{M}}_5(t) = \mathcal{M}_4(t)$ . See Figure 2.27.

In order to answer the question regarding the new control point, we look again at the knot-insertion, now using the previously introduced concept of minimal refinement. Observe that the new knot  $\xi = 3$  lies in the parameter domain of the three original B-splines  $\mathcal{M}_k(t)$  for  $k \in \{0, 1, 2\}$ . This means that their associated tetrahedra  $\mathcal{T}_k$  can each be split into two smaller tetrahedra  $\mathcal{T}_{k,a}$  and  $\mathcal{T}_{k,b}$ , resulting in

<sup>†</sup>For simplicity, we omit the cases where  $\xi$  coincides with an existing knot.



**Figure 2.27:** The supports of the B-splines before knot-insertion (top) and after knot-insertion (bottom). Instead of the parameter domain, we have visualised the supports in the index domain, where knots are equidistant regardless of their values.

$\mathcal{M}_{k,a}(t) + \mathcal{M}_{k,b}(t) = \mathcal{M}_k(t)$  for  $k \in \{0, 1, 2\}$ . Considering the three cases discussed above, we see that

$$\begin{aligned}
 \mathcal{M}_{0,a}(t) : [0, 0, 1, 3], \quad V_{0,a} &= \frac{(3-0)}{(4-0)} \frac{(4-0)}{3} = \frac{(3-0)}{3} = 1, \\
 \mathcal{M}_{0,b}(t) : [0, 1, 3, 4], \quad V_{0,b} &= \frac{(4-3)}{(4-0)} \frac{(4-0)}{3} = \frac{(4-3)}{3} = \frac{1}{3}, \\
 \mathcal{M}_{1,a}(t) : [0, 1, 3, 4], \quad V_{1,a} &= \frac{(3-0)}{(5-0)} \frac{(5-0)}{3} = \frac{(3-0)}{3} = 1, \\
 \mathcal{M}_{1,b}(t) : [1, 3, 4, 5], \quad V_{1,b} &= \frac{(5-3)}{(5-0)} \frac{(5-0)}{3} = \frac{(5-3)}{3} = \frac{2}{3}, \\
 \mathcal{M}_{2,a}(t) : [1, 3, 4, 5], \quad V_{2,a} &= \frac{(3-1)}{(7-1)} \frac{(7-1)}{3} = \frac{(3-1)}{3} = \frac{2}{3}, \\
 \mathcal{M}_{2,b}(t) : [3, 4, 5, 7], \quad V_{2,b} &= \frac{(7-3)}{(7-1)} \frac{(7-1)}{3} = \frac{(7-3)}{3} = \frac{4}{3}.
 \end{aligned}$$

It follows that  $\mathcal{M}_{0,a}(t)$  and  $\mathcal{M}_{2,b}(t)$  are the B-splines  $\bar{\mathcal{M}}_0(t)$  and  $\bar{\mathcal{M}}_3(t)$ , respectively, whereas the others are *scaled* B-splines. In fact,  $\mathcal{M}_{0,b}(t)$  and  $\mathcal{M}_{1,a}(t)$  are both scaled versions of the same B-spline,  $\bar{\mathcal{M}}_1(t)$ , whose defining simplex has a volume of  $\frac{4-0}{3} = \frac{4}{3}$ . In the same fashion,  $\mathcal{M}_{1,b}(t)$  and  $\mathcal{M}_{2,a}(t)$  are both scaled versions of  $\bar{\mathcal{M}}_2(t)$ ,

whose simplex has a volume of  $\frac{5-1}{3} = \frac{4}{3}$ . Therefore, we have

$$\begin{aligned}
 C(t) &= \mathcal{M}_0(t)\mathbf{P}_0 + \mathcal{M}_1(t)\mathbf{P}_1 + \mathcal{M}_2(t)\mathbf{P}_2 + \mathcal{M}_3(t)\mathbf{P}_3 + \mathcal{M}_4(t)\mathbf{P}_4 \\
 &= \left(\mathcal{M}_{0,a}(t) + \mathcal{M}_{0,b}(t)\right)\mathbf{P}_0 + \left(\mathcal{M}_{1,a}(t) + \mathcal{M}_{1,b}(t)\right)\mathbf{P}_1 + \\
 &\quad \left(\mathcal{M}_{2,a}(t) + \mathcal{M}_{2,b}(t)\right)\mathbf{P}_2 + \mathcal{M}_3(t)\mathbf{P}_3 + \mathcal{M}_4(t)\mathbf{P}_4 \\
 &= \left(\bar{\mathcal{M}}_0(t) + \frac{1}{4}\bar{\mathcal{M}}_1(t)\right)\mathbf{P}_0 + \left(\frac{3}{4}\bar{\mathcal{M}}_1(t) + \frac{1}{2}\bar{\mathcal{M}}_2(t)\right)\mathbf{P}_1 + \\
 &\quad \left(\frac{1}{2}\bar{\mathcal{M}}_2(t) + \bar{\mathcal{M}}_3(t)\right)\mathbf{P}_2 + \bar{\mathcal{M}}_4(t)\mathbf{P}_3 + \bar{\mathcal{M}}_5(t)\mathbf{P}_4 \\
 &= \bar{\mathcal{M}}_0(t)\mathbf{P}_0 + \bar{\mathcal{M}}_1(t)\left(\frac{1}{4}\mathbf{P}_0 + \frac{3}{4}\mathbf{P}_1\right) + \bar{\mathcal{M}}_2(t)\left(\frac{1}{2}\mathbf{P}_1 + \frac{1}{2}\mathbf{P}_2\right) + \\
 &\quad \bar{\mathcal{M}}_3(t)\mathbf{P}_2 + \bar{\mathcal{M}}_4(t)\mathbf{P}_3 + \bar{\mathcal{M}}_5(t)\mathbf{P}_4.
 \end{aligned}$$

The resulting control polygon is shown in Figure 2.8, where the grey points are the two new control points, effectively replacing  $\mathbf{P}_1$ . The designer now has more DoFs to locally modify the shape of the curve.

A general formula for knot-insertion for quadratic B-splines readily follows and can be extended to B-splines of arbitrary degree  $d$ .

Summarising, the difference between the two approaches is that knot-insertion can be applied to either a single B-spline (the knot is inserted in a *local* knot-vector, affecting only one function), or to a set of B-splines (the knot is inserted in the *global* knot-vector, generally affecting multiple functions). Only the latter approach results in new control points; in the former approach, the existing control point is now associated with both  $\mathcal{M}_a(t)$  and  $\mathcal{M}_b(t)$ <sup>†</sup>. Insertion in a local knot-vector is often used in numerical methods to refine an approximation space; insertion in the global knot-vector is also used in modelling.

### 2.3.2 Two-scale relation

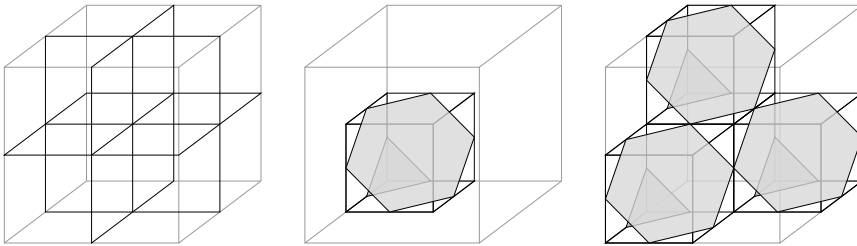
Motivated by the geometrical interpretation of knot-insertion, one might wonder if a similar approach can be applied to box splines. Although we can certainly split unit hypercubes into two and project the individual parts (recall that half-box splines are based on this principle), the results might not be what we are looking for. Instead, we can take a different approach and subdivide each *edge* of a unit hypercube in  $m$  equal parts. This effectively subdivides a hypercube in  $\mathbb{R}^d$  in  $m^d$  smaller hypercubes. The important observation here is that each small hypercube projects to a dilated version of the original box spline  $\mathcal{M}_\Xi$ . The support of these dilated versions scales by a factor  $\frac{1}{m}$  compared to the support of the original box spline, whereas their value scales by a factor  $(\frac{1}{m})^{d-\dim(\text{supp}(\mathcal{M}_\Xi))}$ . Due to the different locations of the smaller hypercubes in the original hypercube, the resulting dilated versions of  $\mathcal{M}_\Xi$  are shifted by combinations of the direction vectors, also scaled by  $\frac{1}{m}$ . In other words, *a box spline can be written as a sum of shifted, dilated versions of itself*. This concept is known as the *two-scale*

<sup>†</sup> An alternative way to interpret this case is that the existing control point is cloned, each copy associated with only one function.

relation [ML95]; usually,  $m = 2$  is assumed, though it holds for general  $m \in \mathbb{N}$ . Below, we illustrate the concept from two different perspectives.

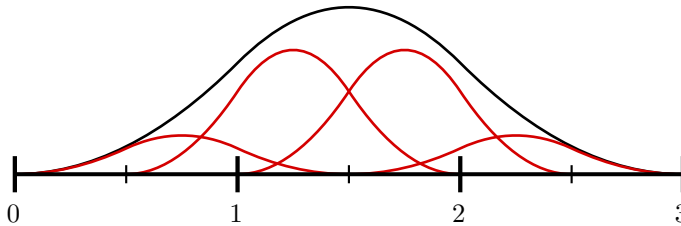
### Projection approach

Considering the unit cube and its projection onto  $\mathbb{R}$ , recall that – when using the direction matrix  $\Xi = (1 \ 1 \ 1)$  – the result is the quadratic uniform B-spline  $\mathcal{M}^2(t)$  (see Figure 2.17). Bisecting each edge of the cube results in  $2^3 = 8$  smaller cubes, each with a volume of  $\frac{1}{8}$ . Note that some of the scaled cubes project to the *same* shifted, dilated version of  $\mathcal{M}^2(t)$ ; see Figure 2.28.



**Figure 2.28:** Bisecting the edges of the cube results in 8 smaller cubes. Groups of small cubes project to the same shifted, dilated version of the original box spline.

As  $\dim(\text{supp}(\mathcal{M}^2)) = 1$ , it follows that  $\mathcal{M}^2(t) = \frac{1}{4} \left( \mathcal{M}^2(2t) + 3\mathcal{M}^2(2t - 1) + 3\mathcal{M}^2(2t - 2) + \mathcal{M}^2(2t - 3) \right)$ . Figure 2.29 illustrates the relation.



**Figure 2.29:** The uniform quadratic B-spline  $\mathcal{M}^2(t)$  (black) can be expressed as the sum of shifted, dilated versions of itself (red).

### Fourier approach

Given the unit pulse  $h(t)$  of unit length centred around  $t = 0$ , its Fourier transform is

$$\hat{h}(\omega) = \frac{\sin(\omega/2)}{\omega/2}, \quad (2.27)$$

with  $\hat{h}(0) = 1$ . Shifting  $h(t)$  to the right half a unit then yields  $f(t) = \mathcal{M}^0(t)$ , the constant B-spline. Its Fourier transform is  $\hat{h}(\omega)$  multiplied by  $e^{-i\omega/2}$  (because of the



shift), resulting in

$$\hat{\mathcal{M}}^0(\omega) = e^{-i\omega/2} \frac{\sin(\omega/2)}{\omega/2} = e^{-i\omega/2} \frac{e^{i\omega/2} - e^{-i\omega/2}}{2i\omega/2} = \frac{1 - e^{-i\omega}}{i\omega}. \quad (2.28)$$

As illustrated, higher-order uniform B-splines can be defined using convolution. As the Fourier transform of a convolution is equivalent to the multiplication of the individual Fourier transforms, this yields

$$\hat{\mathcal{M}}^d(\omega) = \left( \frac{1 - e^{-i\omega}}{i\omega} \right)^{d+1}. \quad (2.29)$$

Now, note that we can write  $\hat{\mathcal{M}}^d(\omega)$  as

$$\begin{aligned} \hat{\mathcal{M}}^d(\omega) &= \left( \frac{1 - e^{-i\omega}}{i\omega} \right)^{d+1} = \left( \frac{1 + e^{-i\omega/2}}{2} \frac{1 - e^{-i\omega/2}}{i\omega/2} \right)^{d+1} \\ &= 2 \left( \frac{1 + e^{-i\omega/2}}{2} \right)^{d+1} \hat{\mathcal{M}}^d(\omega/2). \end{aligned} \quad (2.30)$$

Taking  $\mathcal{M}^2(t)$  as example, we obtain

$$\begin{aligned} \hat{\mathcal{M}}^2(\omega) &= 2 \left( \frac{1 + e^{-i\omega/2}}{2} \right)^3 \hat{\mathcal{M}}^2(\omega/2) \\ &= \frac{1}{4} \left( 1 + 3e^{-i\omega/2} + 3e^{-i\omega} + e^{-3i\omega/2} \right) \hat{\mathcal{M}}^2(\omega/2), \end{aligned} \quad (2.31)$$

which is the Fourier transform of the sum of shifts of dilated versions of  $\mathcal{M}^2(t)$ . The coefficients in the brackets,  $[1, 3, 3, 1]$ , match those we found using the projection approach above. The method extends to general box splines, using shifts in the direction of the arrows.

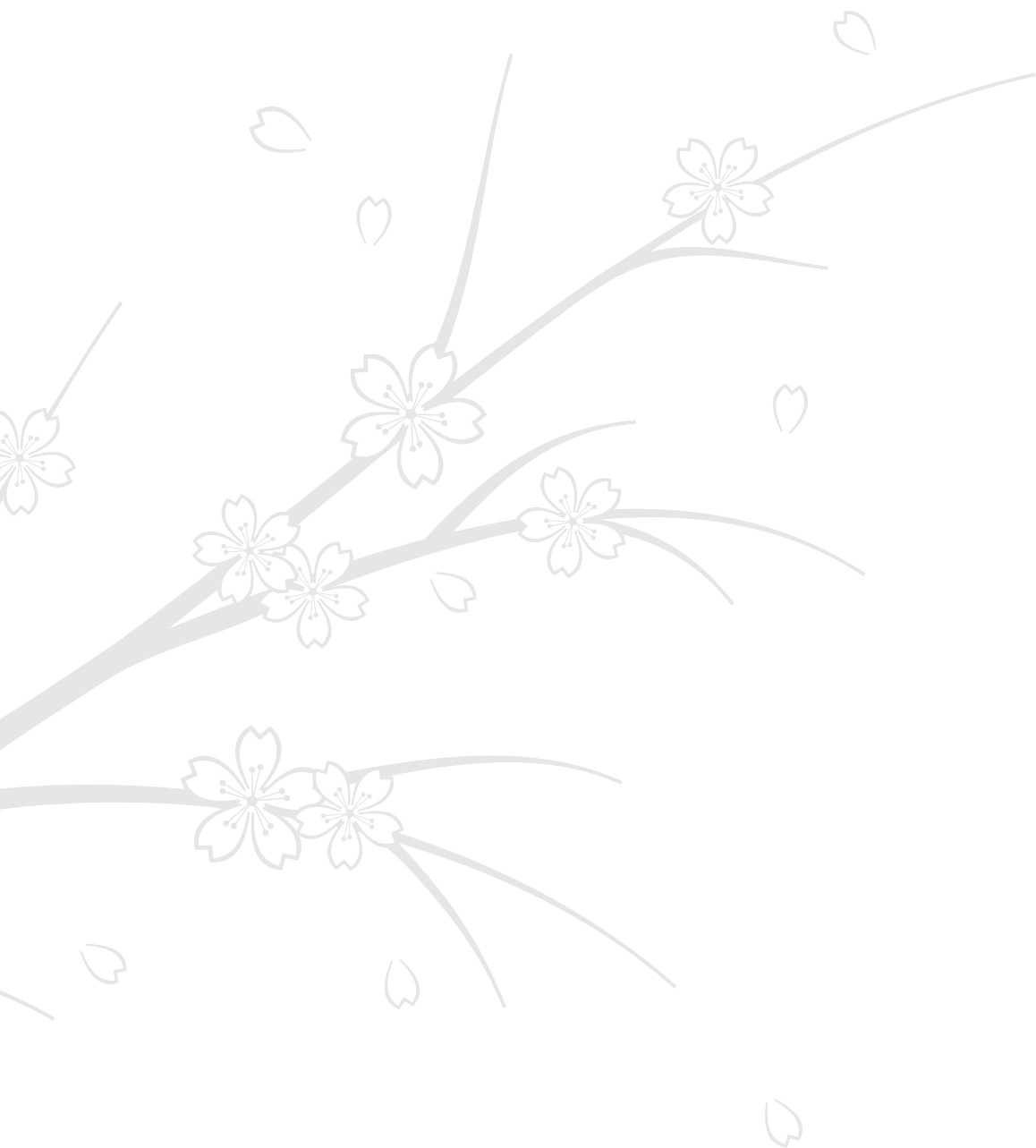


### 3 | Constructing smooth surfaces of arbitrary manifold topology



Parts of this chapter have been published as

- Pieter J Barendrecht, Malcolm A Sabin and Jiří Kosinka. “A bivariate  $C^1$  subdivision scheme based on cubic half-box splines”. In: *Computer Aided Geometric Design* 71 (2019), pp. 77–89. See Section [3.2.1](#).



Given the infinite variety of 3D shapes, the possibility to model objects as smooth surfaces of arbitrary topology is indispensable for designers. In this chapter we take a detailed look at composite  $G^1$  Bézier surfaces and subdivision surfaces, two popular approaches to construct such objects. We conclude with a brief description of selected alternative methods.

## 3.1 $G^1$ composite surfaces

Soon after the discovery of Bézier patches, the quest on how to construct composite  $G^1$  Bézier surfaces began. Many approaches have been explored [Wij86; Sar87; Pip87; SS87; Du88; Wat88; LH89; Deg90; Pet90a; Pet91b; Loo94]; nowadays this direction of research appears to be mostly complete [Pet02; Kic16; BM17]. In this section, we discuss the interpolation of *cubic curve networks* by Bézier patches. In addition, we consider *Gregory patches* [Gre74; CK83] as a possible alternative.

### 3.1.1 Interpolating curve networks with Bézier patches

Assume we are given a quad-triangle mesh of arbitrary manifold topology, possibly with boundaries. Instead of edges connecting the vertices in the mesh, we have cubic Bézier curves connecting the vertices such that the tangent vectors of the curves meeting at a common vertex all lie in a common tangent plane. In other words, the vertices of the mesh act as the first and last control points of the cubic Bézier curves. The two interior control points of each curve — controlling its tangent vectors — are provided<sup>†</sup> in addition to the mesh and do *not* coincide with its vertices. We refer to such a structure as a *cubic curve network*.

Our aim is to interpolate cubic curve networks using quadrilateral and triangular Bézier patches, one patch per original quad or triangle in the mesh, such that the overall result is  $G^1$ . This means that for any point on a curve  $\Gamma$  shared by two patches, the three tangent vectors — the *transversal tangent* of both patches (also referred to as the cross-boundary derivative) and the *versal tangent* along the shared curve  $\Gamma$  — should be co-planar. As such, the first step is to find expressions for the transversal tangent at the boundaries of quadrilateral and triangular patches. In the former case, the expression for a patch of bi-degree  $d$  follows from (2.8) and the tensor product, resulting in

$$\begin{aligned} \left. \frac{\partial S(u, v)}{\partial u} \right|_{u=0} &= d \sum_{l=0}^d \mathcal{B}_l^d(v) \left( \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(u) (\mathbf{P}_{k+1,l} - \mathbf{P}_{k,l}) \right) \Big|_{u=0} \\ &= d \sum_{l=0}^d \mathcal{B}_l^d(v) (\mathbf{P}_{1,l} - \mathbf{P}_{0,l}). \end{aligned} \quad (3.1)$$

The triangular case is more complicated, as there is not a single tangent vector that is logically orthogonal to the patch boundary. Instead, there are two tangent vectors, each aligned to an edge of the parameter domain. Technically, we could choose either one of these aligned tangents to work with. However, if we instead use a combination of them, known as the *radial derivative* [Far82a], we obtain a symmetrical expression that is nicer to work with. Assuming, without loss of generality, that the shared curve  $\Gamma$  is parameterised in the  $v$ -direction, the aligned tangents are the directional derivatives in the barycentric directions  $r_1 = (1, 0, -1)$  and  $r_2 = (0, 1, -1)$ , respectively. From

<sup>†</sup>The interior control points can either be provided manually or be defined by the projection of points on the original edge to the vertex' tangent plane (cf. PN triangles [Vla+01]).

(2.19) it then follows that for a degree  $d$  triangular patch we have

$$\begin{aligned} D_{r_1}T(u, v, w)|_{w=0} &= d \sum_{k+l+m=d-1} \mathcal{B}_{klm}(u, v, w)(\mathbf{P}_{k+1,l,m} - \mathbf{P}_{k,l,m+1}) \Big|_{w=0} \\ &= d \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(v)(\mathbf{P}_{d-k,k,0} - \mathbf{P}_{d-k-1,k,1}). \end{aligned} \quad (3.2)$$

This second step works because all  $\mathcal{B}_{klm}(u, v, w)$  with  $m \neq 0$  vanish on the  $w = 0$  boundary curve, leaving only the basis functions associated with the control points defining this boundary curve of the patch. Recall that these are defined as in (2.17), so that e.g. for quadratics we have  $u^2$ ,  $2uv$  and  $v^2$ . As  $u + v = 1$  on this boundary curve, we can set  $u = (1 - v)$ , resulting in the functions  $(1 - v)^2$ ,  $2(1 - v)v$  and  $v^2$ . These are the univariate quadratic Bernsteins; evidently, this approach works for any  $d$ . In the same fashion, we obtain

$$D_{r_2}T(u, v, w)|_{w=0} = d \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(v)(\mathbf{P}_{d-k-1,k+1,0} - \mathbf{P}_{d-k-1,k,1}). \quad (3.3)$$

The radial derivative is an affine combination of (3.2) and (3.3), namely

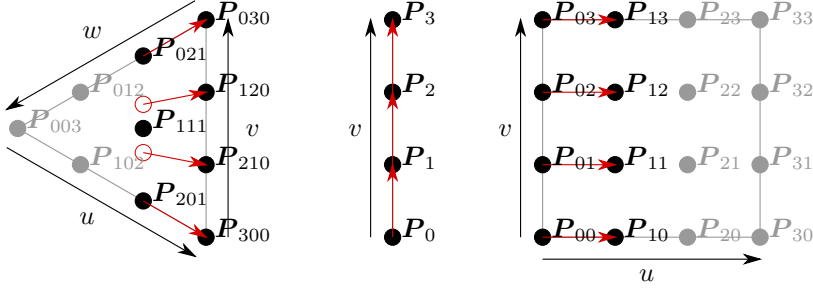
$$\begin{aligned} D_RT(u, v, w)|_{w=0} &= d \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(v) \left( (1 - v)(\mathbf{P}_{d-k,k,0} - \mathbf{P}_{d-k-1,k,1}) + \right. \\ &\quad \left. v(\mathbf{P}_{d-k-1,k+1,0} - \mathbf{P}_{d-k-1,k,1}) \right) \\ &= d \sum_{k=0}^d \mathcal{B}_k^d(v) \left( \mathbf{P}_{d-k,k,0} - \left( \frac{k}{d} \mathbf{P}_{d-k,k-1,1} + \frac{d-k}{d} \mathbf{P}_{d-k-1,k,1} \right) \right). \end{aligned} \quad (3.4)$$

The versal tangent of  $\Gamma(v)$  is simply its derivative (2.8), which we repeat here for convenience;

$$\Gamma'(v) = 3 \sum_{k=0}^2 \mathcal{B}_k^2(v)(\mathbf{P}_{k+1} - \mathbf{P}_k). \quad (3.5)$$

Figure 3.1 summarises the layout of the control points for the different patches and the shared curve.

With the expressions for all tangents given, we introduce the shorthand notation  $T_L(v)$  for the transversal tangent of the patch on the *left* side of the shared curve  $\Gamma(v)$  when traversing it in increasing  $v$  direction, and  $T_R(v)$  for the transversal tangent of the patch on the *right* side of  $\Gamma(v)$ . A patch is either a tensor-product Bézier patch or a triangular Bézier patch, so both  $T_L(v)$  and  $T_R(v)$  can either refer to the transversal tangent of (3.1) or to the radial tangent of (3.4). The  $G^1$  requirement then translates to



**Figure 3.1:** A cubic triangular Bézier patch  $T(u, v, w)$  (left), the shared cubic curve  $\Gamma(v)$  (middle) and a bicubic tensor-product Bézier patch  $S(u, v)$  (right).

the following condition<sup>†</sup>,

$$\det(\mathbf{T}_L(v), \Gamma'(v), \mathbf{T}_R(v)) = 0 \quad \forall v \in [0, 1] \text{ for all curves } \Gamma. \quad (3.6)$$

Unfortunately, this expression is generally understood not to be very constructive for determining the inner control points of the patches such that the connections are everywhere  $G^1$  [Liu86; DeR90]. An alternative expression is

$$\alpha(v)\mathbf{T}_L(v) + \gamma(v)\Gamma'(v) = \beta(v)\mathbf{T}_R(v) \quad \forall v \in [0, 1] \text{ for all curves } \Gamma \quad (3.7)$$

for some unknown coefficient functions  $\alpha(v)$ ,  $\beta(v)$  and  $\gamma(v)$ , which we assume to be polynomial<sup>‡</sup>. The idea is to express the coefficient functions in BB-form, such that the individual terms in (3.7) can be written as combinations of higher-order Bernsteins [DS88; DS90; DS91].

For now, let us focus on quadrilateral patches only, with the additional assumption that all patches are of the same bi-degree  $d = 3$ . To start with, we assume  $\alpha(v)$  and  $\beta(v)$  to be linear. In that case,  $\gamma(v)$  can be quadratic, as both  $\mathbf{T}_L(v)$  and  $\mathbf{T}_R(v)$  are of degree  $d = 3$ , whereas  $\Gamma'(v)$  is of degree 2. We thus have, for some  $\alpha_k, \beta_k, \gamma_k \in \mathbb{R}$ ,

$$\begin{aligned} \alpha(v) &= (1 - v)\alpha_0 + v\alpha_1, \\ \beta(v) &= (1 - v)\beta_0 + v\beta_1, \\ \gamma(v) &= (1 - v)^2\gamma_0 + 2(1 - v)v\gamma_1 + v^2\gamma_2. \end{aligned} \quad (3.8)$$

Note that we can already determine the values of all coefficients except  $\gamma_1$  by evaluating (3.7) at  $v = 0$  and  $v = 1$ , which correspond to the endpoints of the curves  $\Gamma$ . Here, the tangents are simply the differences of the second and first-, or fourth and third control points (scaled by a factor 3) of the different curves. Clearly, these *given* tangent vectors are co-planar by definition of the cubic curve network. It is also the reason why  $\alpha(v)$  and  $\beta(v)$  can, in general, not be constant in this context. Now, observe that at both endpoints we have one scalar degree of freedom (DoF); we choose to

<sup>†</sup>To be precise, this is only the main condition. There are three additional conditions — the patches should connect  $G^0$ , should be oriented properly to avoid cusps, and have well-defined normals  $\mathbf{T}_L(v) \times \Gamma'(v) \neq 0$  along the shared curve.

<sup>‡</sup>Note that we could also choose to use *rational* functions, e.g.  $\alpha(v)/\beta(v)$ ,  $\gamma(v)/\beta(v)$  and 1, though this takes out some of the symmetry of the problem.

set  $\beta_0 = 1$  and  $\alpha_1 = 1$ . Then, the remaining coefficients (except  $\gamma_1$ ) are obtained by solving

$$\begin{pmatrix} \mathbf{T}_L(0).x & \Gamma'(0).x \\ \mathbf{T}_L(0).y & \Gamma'(0).y \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \gamma_0 \end{pmatrix} = \begin{pmatrix} \mathbf{T}_R(0).x \\ \mathbf{T}_R(0).y \end{pmatrix}, \quad (3.9)$$

with  $(\cdot).x$  and  $(\cdot).y$  the  $x$ - and  $y$ -components of the tangent vectors. Naturally, they also have a  $z$ -component, but two components are sufficient to solve this system<sup>†</sup>.

Instead of constructing another  $2 \times 2$  system to solve for  $\beta_1$  and  $\gamma_2$ , we can consider the shared curve and the patches from the opposite direction (i.e. reverse the parametrization in  $v$ ). It follows that our unknowns  $\beta_1$  and  $\gamma_2$  become the unknowns  $\alpha_0$  and  $\gamma_0$  in this reversed orientation, respectively. Moreover,  $\alpha_1$  becomes  $\beta_0$ , which is in fact the reason we chose to set both to 1. The system to compute  $\beta_1$  and  $\gamma_2$  thus looks identical to (3.9), but note that e.g.  $\mathbf{T}_L(0)$  in this orientation refers to  $\mathbf{T}_R(1)$  in the original orientation. In other words, we can compute all coefficients (except for the  $\gamma_1$ ) for the entire cubic curve network by looping over all vertices and solving (3.9) for all curves connected to each vertex. It also shows that the problem is symmetric, i.e. it does not matter which orientation we consider.

We now substitute (3.8) into (3.7), and obtain

$$\begin{aligned} \alpha(v)\mathbf{T}_L(v) &= 3((1-v)\alpha_0 + v\alpha_1) \sum_{k=0}^3 \mathcal{B}_k^3(v) (\mathbf{P}_k - \mathbf{P}_{2,k}^L) \\ &= 3 \sum_{k=0}^4 \mathcal{B}_k^4(v) \left( \frac{4-k}{4} \alpha_0 (\mathbf{P}_k - \mathbf{P}_{2,k}^L) + \frac{k}{4} \alpha_1 (\mathbf{P}_{k-1} - \mathbf{P}_{2,k-1}^L) \right), \\ \beta(v)\mathbf{T}_R(v) &= 3 \sum_{k=0}^4 \mathcal{B}_k^4(v) \left( \frac{4-k}{4} \beta_0 (\mathbf{P}_{1,k}^R - \mathbf{P}_k) + \frac{k}{4} \beta_1 (\mathbf{P}_{1,k-1}^R - \mathbf{P}_{k-1}) \right), \\ \gamma(v)\Gamma'(v) &= 3 \sum_{k=0}^4 \mathcal{B}_k^4(v) \left( \frac{(3-k)(4-k)}{12} \gamma_0 (\mathbf{P}_{k+1} - \mathbf{P}_k) + \right. \\ &\quad \left. \frac{2k(4-k)}{12} \gamma_1 (\mathbf{P}_k - \mathbf{P}_{k-1}) + \frac{k(k-1)}{12} \gamma_2 (\mathbf{P}_{k-1} - \mathbf{P}_{k-2}) \right), \end{aligned}$$

where  $\mathbf{P}_{2,k}^L$  refers to the control point immediately to the left of  $\mathbf{P}_k$ , and  $\mathbf{P}_{1,k}^R$  the control point immediately to the right (see Figure 3.1). Additionally, note that the indices of the control points go out of range, though this only happens when they are multiplied by 0; it can therefore be safely ignored. Rewriting (3.7) using the above results

<sup>†</sup>The  $z$ -component could be added, resulting in a  $3 \times 2$  matrix and  $3 \times 1$  right-hand side, making (3.9) overdetermined. Theoretically this does not affect the solution due to co-linearity, but numerically, it might.



in

$$\begin{aligned} \sum_{k=0}^4 & \left( \frac{4-k}{4} \alpha_0 (\mathbf{P}_k - \mathbf{P}_{2,k}^L) + \frac{k}{4} \alpha_1 (\mathbf{P}_{k-1} - \mathbf{P}_{2,k-1}^L) + \right. \\ & \frac{(3-k)(4-k)}{12} \gamma_0 (\mathbf{P}_{k+1} - \mathbf{P}_k) + \frac{2k(4-k)}{12} \gamma_1 (\mathbf{P}_k - \mathbf{P}_{k-1}) + \\ & \frac{k(k-1)}{12} \gamma_2 (\mathbf{P}_{k-1} - \mathbf{P}_{k-2}) - \\ & \left. \frac{4-k}{4} \beta_0 (\mathbf{P}_{1,k}^R - \mathbf{P}_k) - \frac{k}{4} \beta_1 (\mathbf{P}_{1,k-1}^R - \mathbf{P}_{k-1}) \right) \mathcal{B}_k^4(v) = 0. \end{aligned} \quad (3.10)$$

Recall that the Bernsteins form a basis, which implies that the term for each  $k$  in (3.10) should vanish. We thus end up with  $d+2 = 5$  *vectorial* equations. For example, for  $k = 0$  we have

$$\alpha_0 (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) + \gamma_0 (\mathbf{P}_1 - \mathbf{P}_0) - \beta_0 (\mathbf{P}_{1,0}^R - \mathbf{P}_0) = 0, \quad (3.11)$$

which is equivalent to (3.9); for  $k = 4$  we have a similar (symmetric) result.

We now focus on determining the interior control points of the patch. First, we consider  $k = 1$  (or by symmetry,  $k = 3$ ) in (3.10), yielding

$$\begin{aligned} \frac{3}{4} \alpha_0 (\mathbf{P}_1 - \mathbf{P}_{2,1}^L) + \frac{1}{4} \alpha_1 (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) + \frac{1}{2} \gamma_0 (\mathbf{P}_2 - \mathbf{P}_1) + \frac{1}{2} \gamma_1 (\mathbf{P}_1 - \mathbf{P}_0) - \\ \frac{3}{4} \beta_0 (\mathbf{P}_{1,1}^R - \mathbf{P}_1) - \frac{1}{4} \beta_1 (\mathbf{P}_{1,0}^R - \mathbf{P}_0) = 0. \end{aligned} \quad (3.12)$$

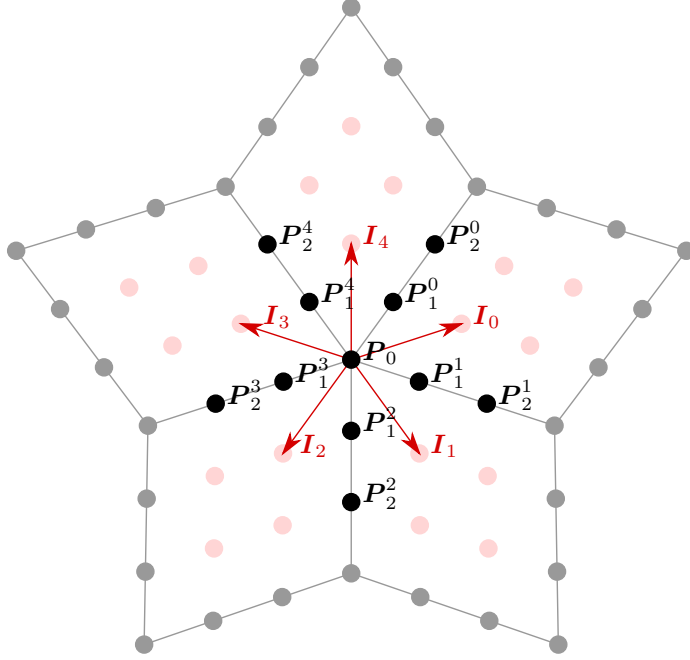
Isolating the unknowns  $\mathbf{P}_{2,1}^L$  and  $\mathbf{P}_{1,1}^R$ , we obtain

$$\begin{aligned} \alpha_0 \mathbf{P}_{2,1}^L + \beta_0 \mathbf{P}_{1,1}^R = \alpha_0 \mathbf{P}_1 + \frac{1}{3} \alpha_1 (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) + \frac{2}{3} \gamma_0 (\mathbf{P}_2 - \mathbf{P}_1) + \\ \frac{2}{3} \gamma_1 (\mathbf{P}_1 - \mathbf{P}_0) + \beta_0 \mathbf{P}_1 + \frac{1}{3} \beta_1 (\mathbf{P}_0 - \mathbf{P}_{1,0}^R). \end{aligned}$$

From a geometrical point of view, this is not a very clean expression – in general  $\alpha_0 \neq 0$  (and therefore  $\alpha_0 + \beta_0 \neq 1$ ), which means that the linear combinations of the isolated control points are in general not affine combinations. We can improve this by subtracting a fixed control point from all isolated control points; choosing  $\mathbf{P}_0$  results in

$$\begin{aligned} \alpha_0 (\mathbf{P}_{2,1}^L - \mathbf{P}_0) + \beta_0 (\mathbf{P}_{1,1}^R - \mathbf{P}_0) = \alpha_0 (\mathbf{P}_1 - \mathbf{P}_0) + \frac{1}{3} \alpha_1 (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) + \\ \frac{2}{3} \gamma_0 (\mathbf{P}_2 - \mathbf{P}_1) + \frac{2}{3} \gamma_1 (\mathbf{P}_1 - \mathbf{P}_0) + \beta_0 (\mathbf{P}_1 - \mathbf{P}_0) + \frac{1}{3} \beta_1 (\mathbf{P}_0 - \mathbf{P}_{1,0}^R). \end{aligned} \quad (3.13)$$

At first sight, with two unknowns and only one equation, it might seem as though we have a vectorial DoF here. However, this is not the case; each interior control



**Figure 3.2:** The unknown vectors  $I_l$  defining the interior control points are each shared among three patches, which leads to a cyclic system for solving them.

point is actually shared among *three* patches<sup>†</sup>, which ultimately results in a cyclic system [Wij86; DS88; Pet91b]. See Figure 3.2.

Again, we introduce new notation for certain control points and vectors. The  $n$  curves connected to the central vertex  $P_0$  get subscripts  $l \in [0, n - 1]$ , resulting in  $\Gamma_l(v)$ . The control points defining these curves get the same indices  $l$  as superscripts, e.g.  $P_1^l$  and  $P_2^l$ . Observe that for e.g.  $l = 1$  we then have that  $P_{2,0}^L = P_1^0$  and  $P_{1,0}^R = P_1^2$ . The unknowns become  $(P_{2,1}^L - P_0) = I_0$  and  $(P_{1,1}^R - P_0) = I_1$ . Finally, for the right-hand side of (3.13) we use the shorthand notation  $F_l$ . Ultimately, for the example shown in Figure 3.2 we obtain the system

$$\begin{pmatrix} \beta_0^0 & 0 & 0 & 0 & \alpha_0^0 \\ \alpha_0^1 & \beta_0^1 & 0 & 0 & 0 \\ 0 & \alpha_0^2 & \beta_0^2 & 0 & 0 \\ 0 & 0 & \alpha_0^3 & \beta_0^3 & 0 \\ 0 & 0 & 0 & \alpha_0^4 & \beta_0^4 \end{pmatrix} \begin{pmatrix} I_0 \\ I_1 \\ I_2 \\ I_3 \\ I_4 \end{pmatrix} = \begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix}. \quad (3.14)$$

For general  $n$  (i.e. the valency of the central point), we denote this matrix as  $M_n$ . As it turns out, this matrix is singular for even  $n$  [DS88]. To prove this, we express its

<sup>†</sup>Boundary- and corner patches are an exception — we will briefly touch upon these towards the end of this section.

determinant using co-factor expansion along the first row:

$$\begin{aligned} \det(M_n) &= \beta_0^0 \det \begin{pmatrix} \beta_0^1 & 0 & \dots & 0 \\ \alpha_0^2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ 0 & 0 & \alpha_0^{n-1} & \beta_0^{n-1} \end{pmatrix} + (-1)^{1+n} \alpha_0^0 \det \begin{pmatrix} \alpha_0^1 & \beta_0^1 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \beta_0^{n-2} \\ 0 & \dots & 0 & \alpha_0^{n-1} \end{pmatrix} \\ &= \beta_0^0 \prod_{k=1}^{n-1} \beta_0^k + (-1)^{1+n} \alpha_0^0 \prod_{k=1}^{n-1} \alpha_0^k = \prod_{k=0}^{n-1} \beta_0^k + (-1)^{1+n} \prod_{k=0}^{n-1} \alpha_0^k. \end{aligned} \quad (3.15)$$

For the next step in this proof, observe that if we take the cross-product of (3.11) with  $(P_1 - P_0)$ , we obtain [DS88]

$$\alpha_0(P_0 - P_{2,0}^L) \times (P_1 - P_0) - \beta_0(P_{1,0}^R - P_0) \times (P_1 - P_0) = 0,$$

and thus

$$\frac{\alpha_0}{\beta_0} = \frac{(P_{1,0}^R - P_0) \times (P_1 - P_0)}{(P_0 - P_{2,0}^L) \times (P_1 - P_0)} = \frac{(P_{1,0}^R - P_0) \times (P_1 - P_0)}{(P_1 - P_0) \times (P_{2,0}^L - P_0)}.$$

Then, using the notation from Figure 3.2, we get

$$\prod_{k=0}^{n-1} \frac{\alpha_0^k}{\beta_0^k} = \prod_{k=0}^{n-1} \frac{(P_1^{k+1} - P_0) \times (P_1^k - P_0)}{(P_1^k - P_0) \times (P_1^{k-1} - P_0)} = 1,$$

with  $k$  understood modulo  $n$ . Therefore,

$$\prod_{k=0}^{n-1} \alpha_0^k = \prod_{k=0}^{n-1} \beta_0^k, \quad (3.16)$$

which – together with (3.15) – proves  $\det(M_n) = 0$  for even valencies.  $\square$

Let us look at an example of  $n$  even, say,  $n = 4$ . Because of the structure of the matrix, it is not too difficult to see that it has rank  $n-1 = 3$ . Applying matrix operations to the *augmented matrix*  $(M_4|F)$  with  $R_k$  indicating the  $k^{th}$  row of  $M_4$  results in

$$\begin{aligned} \left( \begin{array}{cccc|c} \beta_0^0 & 0 & 0 & \alpha_0^0 & F_0 \\ \alpha_0^1 & \beta_0^1 & 0 & 0 & F_1 \\ 0 & \alpha_0^2 & \beta_0^2 & 0 & F_2 \\ 0 & 0 & \alpha_0^3 & \beta_0^3 & F_3 \end{array} \right) &\xrightarrow{R_4 - \frac{\beta_0^3}{\alpha_0^0} R_1} \left( \begin{array}{cccc|c} \beta_0^0 & 0 & 0 & \alpha_0^0 & F_0 \\ \alpha_0^1 & \beta_0^1 & 0 & 0 & F_1 \\ 0 & \alpha_0^2 & \beta_0^2 & 0 & F_2 \\ -\frac{\beta_0^3}{\alpha_0^0} \beta_0^0 & 0 & \alpha_0^3 & 0 & F_3 - \frac{\beta_0^3}{\alpha_0^0} F_0 \end{array} \right) &\xrightarrow{R_4 + \frac{\beta_0^3 \beta_0^0}{\alpha_0^0 \alpha_0^1} R_2} \\ \left( \begin{array}{cccc|c} \beta_0^0 & 0 & 0 & \alpha_0^0 & F_0 \\ \alpha_0^1 & \beta_0^1 & 0 & 0 & F_1 \\ 0 & \alpha_0^2 & \beta_0^2 & 0 & F_2 \\ 0 & \frac{\beta_0^3 \beta_0^0}{\alpha_0^0 \alpha_0^1} \beta_0^1 & \alpha_0^3 & 0 & F_3 - \frac{\beta_0^3}{\alpha_0^0} F_0 + \frac{\beta_0^3 \beta_0^0}{\alpha_0^0 \alpha_0^1} F_1 \end{array} \right) &\xrightarrow{R_4 - \frac{\beta_0^3 \beta_0^0 \beta_0^1}{\alpha_0^0 \alpha_0^1 \alpha_0^2} R_3} \\ \left( \begin{array}{cccc|c} \beta_0^0 & 0 & 0 & \alpha_0^0 & F_0 \\ \alpha_0^1 & \beta_0^1 & 0 & 0 & F_1 \\ 0 & \alpha_0^2 & \beta_0^2 & 0 & F_2 \\ 0 & 0 & 0 & 0 & F_3 - \frac{\beta_0^3}{\alpha_0^0} F_0 + \frac{\beta_0^3 \beta_0^0}{\alpha_0^0 \alpha_0^1} F_1 - \frac{\beta_0^3 \beta_0^0 \beta_0^1}{\alpha_0^0 \alpha_0^1 \alpha_0^2} F_2 \end{array} \right). \end{aligned}$$

In other words, and generalising for arbitrary even  $n \in \mathbb{Z}$ , the right-hand side of (3.14) is therefore only in the column space of  $\mathbf{M}_n$  if

$$\sum_{j=0}^{n-1} (-1)^{j+1} \left( \prod_{k=0}^j \frac{\beta_0^{k-1}}{\alpha_0^k} \right) \mathbf{F}_j = 0, \quad (3.17)$$

again with  $k$  understood modulo  $n$ . This is known as the *vertex enclosure constraint* (VEC) or *compatibility equation*, and is an additional constraint that has to be satisfied for even  $n$  in order for a  $G^1$  connection to be possible.

We note that there is an alternative method to arrive at (3.17), which follows from the observation that the mixed second-order partial derivatives of neighbouring Bézier patches should match at shared corners [Wat88; DS91; Loo94]. Taking the partial derivative of both sides of (3.7) w.r.t.  $v$  gives

$$\alpha'(v)\mathbf{T}_L(v) + \alpha(v) \left[ \frac{d\mathbf{T}_L}{dv} \right] + \gamma'(v)\Gamma'(v) + \gamma(v)\Gamma''(v) = \beta'(v)\mathbf{T}_R(v) + \beta(v) \left[ \frac{d\mathbf{T}_R}{dv} \right]. \quad (3.18)$$

Assuming bicubic patches, using the same coefficient functions as in (3.8), and evaluating at the corner  $(u, v) = (0, 0)$  then yields

$$\begin{aligned} & (\alpha_1 - \alpha_0) \left( \mathbf{P}_0 - \mathbf{P}_{2,0}^L \right) + 3\alpha_0 \left[ \left( (\mathbf{P}_1 - \mathbf{P}_{2,1}^L) - (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) \right) \right] + \\ & 2(\gamma_1 - \gamma_0) (\mathbf{P}_1 - \mathbf{P}_0) + 2\gamma_0 \left( (\mathbf{P}_2 - \mathbf{P}_1) - (\mathbf{P}_1 - \mathbf{P}_0) \right) \\ & = (\beta_1 - \beta_0) \left( \mathbf{P}_{1,0}^R - \mathbf{P}_0 \right) + 3\beta_0 \left[ \left( (\mathbf{P}_{1,1}^R - \mathbf{P}_1) - (\mathbf{P}_{1,0}^R - \mathbf{P}_0) \right) \right]. \end{aligned} \quad (3.19)$$

The outlined terms are the mixed partials, or *twist vectors*. Isolating these yields the same matrix as in (3.14), though the resulting system has a subtly different right-hand side. Note, however, that we can subtract a multiple of (3.11) from (3.19),

$$4 \left( \alpha_0 (\mathbf{P}_0 - \mathbf{P}_{2,0}^L) + \gamma_0 (\mathbf{P}_1 - \mathbf{P}_0) - \beta_0 (\mathbf{P}_{1,0}^R - \mathbf{P}_0) \right) = 0,$$

which then results in a multiple of (3.12), making the two equivalent [DS91].

Returning to our running example using bicubic patches, we now stumble upon the following issue — the values of the  $\gamma_1^l$  in the  $\mathbf{F}_l$  are still unknown. Observe that they appear in the cyclic systems on *both* ends of the curve  $\Gamma_l(v)$ . As such, we cannot include them as unknowns in the computation of the  $\mathbf{I}_l$  without ending up with a *global* system computing all interior points for the curve network at once, which is usually undesired. It also leaves us without DoFs to satisfy (3.17) for even  $n$  (more on this below). One option is to assume  $\gamma_l(v)$  to be linear and subsequently degree-elevate it to a quadratic polynomial, thereby obtaining a value for  $\gamma_1^l$ , but this limits the possible outcomes. In addition, we have yet another vectorial equation to satisfy (associated with  $k = 2$ ), see (3.10), without further available DoFs. This shows that the problem is *overdetermined*, and that in general, bicubic patches are not sufficient to  $G^1$  interpolate a cubic curve network.

Our next approach is to consider bi-degree  $d = 4$  patches. Now,  $\mathbf{T}_L(v)$  and  $\mathbf{T}_R(v)$  are both of degree  $d = 4$ , though  $\Gamma'(v)$  is still of degree 2. As such, with  $\alpha(v)$  and  $\beta(v)$  linear,  $\gamma(v)$  can be cubic:

$$\gamma(v) = (1-v)^3\gamma_0 + 3(1-v)^2v\gamma_1 + 3(1-v)v^2\gamma_2 + v^3\gamma_3.$$

Setting  $\beta_0 = 1$  and  $\alpha_1 = 1$ , we can still apply (3.9) to compute  $\alpha_0$  and  $\gamma_0$ , and when reversing the orientation,  $\beta_1$  and  $\gamma_3$ . Only  $\gamma_1$  and  $\gamma_2$  remain unknown. Instead of (3.10), we now end up with

$$\begin{aligned} \sum_{k=0}^5 \left( \frac{5-k}{5} \alpha_0 (\mathbf{Q}_k - \mathbf{P}_{2,k}^L) + \frac{k}{5} \alpha_1 (\mathbf{Q}_{k-1} - \mathbf{P}_{2,k-1}^L) + \right. \\ \frac{3(3-k)(4-k)(5-k)}{240} \gamma_0 (\mathbf{P}_{k+1} - \mathbf{P}_k) + \frac{9k(4-k)(5-k)}{240} \gamma_1 (\mathbf{P}_k - \mathbf{P}_{k-1}) + \\ \frac{9k(k-1)(5-k)}{240} \gamma_2 (\mathbf{P}_{k-1} - \mathbf{P}_{k-2}) + \frac{3k(k-1)(k-2)}{240} \gamma_3 (\mathbf{P}_{k-2} - \mathbf{P}_{k-3}) - \\ \left. \frac{5-k}{5} \beta_0 (\mathbf{P}_{1,k}^R - \mathbf{Q}_k) - \frac{k}{5} \beta_1 (\mathbf{P}_{1,k-1}^R - \mathbf{Q}_{k-1}) \right) \mathcal{B}_k^5(v) = 0, \end{aligned} \quad (3.20)$$

with  $\mathbf{Q}_k$  the  $\mathbf{P}_k$  degree-elevated to quartics. We thus have  $d+2 = 6$  vectorial equations to satisfy for a  $G^1$  connection between two patches. The cases  $k = 0$  and  $k = 5$  are identical to the setting with bicubic patches<sup>†</sup>.

Similarly, as with bicubic patches,  $k = 1$  (and by symmetry,  $k = 4$ ) results in a cyclic system, only with slightly different expressions for the  $\mathbf{F}_l$  in this case. Note that the structure of the matrix  $\mathbf{M}_n$  remains the same regardless of the degree of the patches; the VEC from (3.17) is a general result. An advantage of using quartic patches in combination with a cubic  $\gamma(v)$  is that we now have a separate coefficient for each end of the curve (i.e.  $\gamma_1$  and  $\gamma_2$ ) that can be used as DoFs to potentially satisfy (3.17). Figure 3.3 illustrates this and compares the use of cubic and quartic patches.

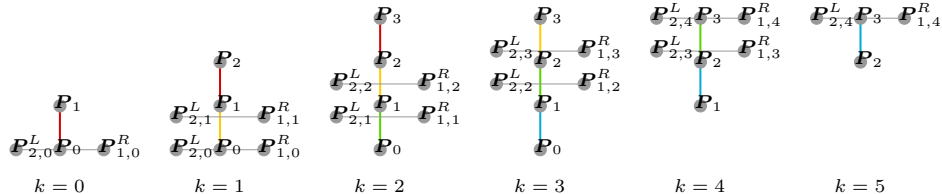
The expression for  $\mathbf{F}_l$  follows from (3.20) as

$$\begin{aligned} \mathbf{F}_l &= \alpha_0^l (\mathbf{Q}_1^l - \mathbf{P}_0) + \frac{1}{4} \alpha_1^l (\mathbf{P}_0 - \mathbf{Q}_1^{l-1}) + \frac{3}{8} \gamma_0^l (\mathbf{P}_2^l - \mathbf{P}_1^l) + \frac{9}{16} \gamma_1^l (\mathbf{P}_1^l - \mathbf{P}_0) + \\ &\quad \beta_0^l (\mathbf{Q}_1^l - \mathbf{P}_0) + \frac{1}{4} \beta_1^l (\mathbf{P}_0 - \mathbf{Q}_1^{l+1}) \\ &= \mathbf{F}_l^k + \frac{9}{16} \gamma_1^l (\mathbf{P}_1^l - \mathbf{P}_0) = \mathbf{F}_l^k + \mathbf{F}_l^u, \end{aligned}$$

with  $\mathbf{F}_l^k$  as shorthand for the *known* terms of  $\mathbf{F}_l$ , and  $\mathbf{F}_l^u$  for the *unknown* terms. Then, writing (3.17) as  $\sum_{j=0}^{n-1} p_j \mathbf{F}_j = 0$ , we obtain

$$\sum_{j=0}^{n-1} p_j \mathbf{F}_j^u = - \sum_{j=0}^{n-1} p_j \mathbf{F}_j^k,$$

<sup>†</sup>Even though we now have quartic patches, the patch boundaries are still only cubic. The  $\mathbf{P}_{2,k}^L$  and  $\mathbf{P}_{1,k}^R$  are now degree-elevated points, and are the reason for the appearance of a factor  $\frac{3}{4}$  for  $k = 0$  and  $k = 5$  in the term with  $\gamma_0$  or  $\gamma_3$ , respectively.



**Figure 3.3:** Diagrams of the control points involved in (3.10) for  $k \in [0, 4]$  (top) and those involved in (3.20) for  $k \in [0, 5]$  (bottom). The vectors  $(\mathbf{P}_l - \mathbf{P}_{l-1})$  are colour-coded, where red corresponds to  $\gamma_0$ , yellow to  $\gamma_1$ , green to  $\gamma_2$  and blue to  $\gamma_3$ . The degree-elevated points  $\mathbf{Q}_l$  have been omitted in the bottom diagram.

which leads to

$$\left( p_0(\mathbf{P}_1^0 - \mathbf{P}_0), p_1(\mathbf{P}_1^1 - \mathbf{P}_0), \dots, p_{n-1}(\mathbf{P}_1^{n-1} - \mathbf{P}_0) \right) \begin{pmatrix} \gamma_1^0 \\ \gamma_1^1 \\ \vdots \\ \gamma_1^{n-1} \end{pmatrix} = -\frac{16}{9} \sum_{j=0}^{n-1} p_j \mathbf{F}_j^{\mathbf{k}}. \quad (3.21)$$

Looking at this component-wise (i.e. in terms of  $x$ -,  $y$ - and  $z$ -components), we have a  $3 \times n$  system, in other words, 3 equations and  $n$  unknowns. For even  $n \geq 4$  we thus have an *underdetermined system*. Although this might sound like good news, observe that all  $(\mathbf{P}_1^j - \mathbf{P}_0)$  are by construction co-planar, which means that  $\sum p_j (\mathbf{P}_1^j - \mathbf{P}_0)$  also lives in the same plane. However,  $\mathbf{F}_j^k$  contains the term  $\frac{3}{8} \gamma_0^j (\mathbf{P}_2^j - \mathbf{P}_1^j)$ , which is typically *not* co-planar with the other vectors. Unfortunately, this means that there are only two cases for which a solution exists:

1. The weighted sum of  $\frac{3}{8}\gamma_0^j p_j(\mathbf{P}_2^j - \mathbf{P}_1^j)$  cancels out the component that is not co-planar. This only happens for surfaces that locally agree with some second fundamental form [Pet91b].
2. All  $\gamma_0^j$  are zero. This only happens for  $n = 4$  when the opposite tangent vectors are pair-wise co-linear [Sar87] (i.e.  $\mathbf{P}_1^m$ ,  $\mathbf{P}_0$  and  $\mathbf{P}_1^{m+2}$  are co-linear for  $m \in \{0, 1\}$ ). This is sometimes referred to as  $X$ -tangents.

Again, this is a general result — neither increasing the degree  $d$  of the patches, nor increasing the degrees of the coefficient functions  $\alpha(v)$ ,  $\beta(v)$  and  $\gamma(v)$  improves

the situation<sup>†</sup>. This means that, when neither of these cases is satisfied, we are out of options for a cubic curve network. A possible workaround would be to degree-elevate the curves to quintics and allow the middle two control points ( $\mathbf{R}_2$  and  $\mathbf{R}_3$ ) to be modified, affecting the original curves.

Still, for a cubic curve network with all interior valencies  $n \in \{3, 4, 5\}$  and  $X$ -tangents (i.e. vanishing  $\gamma_0^l$  and/or  $\gamma_3^l$  coefficients) for all vertices of valency  $n = 4$ , a  $G^1$  interpolation should indeed be possible using biquartics. What is left is a strategy to tune the  $\gamma_1^l$  (and by symmetry  $\gamma_2^l$ ) for *both* odd and even  $n$  [BM17], and solving a  $2 \times 2$  system to satisfy (3.20) for  $k = 2$  and  $k = 3$  simultaneously. Tuning coefficients can be done by (iteratively) optimising different objective functions, e.g. minimising the difference w.r.t. a preferred reference layout of the interior control points. The system for solving the remaining two interior points follows from (3.20),

$$\begin{pmatrix} \alpha_0 & \beta_0 = 1 \\ \alpha_1 = 1 & \beta_1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_{2,2}^L \\ \mathbf{P}_{1,2}^R \end{pmatrix} = \begin{pmatrix} \mathbf{G}_2 \\ \mathbf{G}_3 \end{pmatrix}, \quad (3.22)$$

with shorthand notation  $\mathbf{G}_k$  for the right-hand sides for  $k \in \{2, 3\}$ . This is generally a regular system, unless the tangent vectors associated with the  $\alpha$  and  $\beta$  coefficients are pair-wise mirror-symmetric w.r.t. the tangents associated with the  $\gamma$  coefficients. Note that the matrix can be deduced directly from Figure 3.3 (bottom).

In practice it frequently happens that solutions which are mathematically  $G^1$  (i.e. satisfy the above equations for all  $k$ ) are not visually appealing. As such, more DoFs are required to further tune the solutions. For this reason, bi-degree 5 patches are commonly used [Pet90a; PF10]. The process for biquintics is largely similar to that for biquartics, with the difference that there are now  $2 \times 2$  remaining interior points that are governed by only 3 equations (i.e. associated with  $k \in \{2, 3, 4\}$ ). This provides a vectorial DoF for tuning the outcome. Moreover, a quartic  $\gamma(v)$  also provides a scalar DoF  $\gamma_2$ .

Finally, we generalise the approach to include triangular patches. Assuming a mix of biquartic patches and quartic triangular patches for the moment, observe that with a degree-elevated boundary curve (from cubic to quartic), the degree-elevated control points are defined as

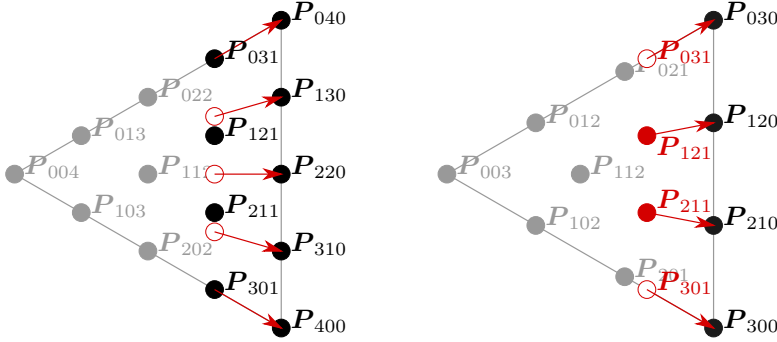
$$\mathbf{Q}_{d-k,k,0} = \frac{k}{d} \mathbf{P}_{d-k,k-1,0} + \frac{d-k}{d} \mathbf{P}_{d-k-1,k,0},$$

<sup>†</sup>Increasing the degree of  $\alpha(v)$  and  $\beta(v)$  to cubics or higher allows the use of coefficients  $\alpha_1$  and  $\beta_1$  as DoFs, though as these are associated with vectors that are co-planar with the others, it does not help in satisfying the VEC.

such that (3.4) can be written as

$$\begin{aligned}
 D_RT(u, v, w)|_{w=0} &= d \sum_{k=0}^d \mathcal{B}_k^d(v) \left( Q_{d-k,k,0} - \left( \frac{k}{d} Q_{d-k,k-1,1} + \frac{d-k}{d} Q_{d-k-1,k,1} \right) \right) \\
 &= d \sum_{k=0}^d \mathcal{B}_k^d(v) \left( \frac{k}{d} (P_{d-k,k-1,0} - Q_{d-k,k-1,1}) + \right. \\
 &\quad \left. \frac{d-k}{d} (P_{d-k-1,k,0} - Q_{d-k-1,k,1}) \right) \\
 &= d \sum_{k=0}^{d-1} \mathcal{B}_k^{d-1}(v) (P_{d-k-1,k,0} - Q_{d-k-1,k,1}). \tag{3.23}
 \end{aligned}$$

Note that the last expression is very similar to the transversal derivative for a tensor-product patch as given in (3.1). Figure 3.4 illustrates the simplification.



**Figure 3.4:** Radial derivative of a quartic triangular patch (left). If the boundary curves are cubics, the radial derivative simplifies (right).

Quartic triangles are generally unsuitable for the same reasons bicubic patches are. Going up a degree, quintic triangular patches combine nicely with biquartic patches – both have quartic transversal derivatives and three interior control points. Likewise, sextic triangles combine nicely with biquintic patches, and provide more DoFs for tuning the positions of control points [Loo94]. The required adaptations for plugging triangles into (3.7) are minimal<sup>†</sup>.

So far, we have not considered boundary patches (including corner patches). These are actually easier to deal with than interior patches, as for boundary patches there is no cyclic system (and therefore no VEC) to satisfy at the boundary. As such, there are more DoFs to tweak the interior points.

For completeness, we remark that the degrees of  $\alpha(v)$ ,  $\beta(v)$  and  $\gamma(v)$  are bounded.

<sup>†</sup>It is entirely possible to combine quintic triangles with biquintic patches, or sextic triangles with bi-quartic patches. However, due to the different degrees of the transversal derivatives (and therefore different degrees of  $\alpha(v)$  and  $\beta(v)$ ) and different numbers of interior control points, the changes are more significant.



This follows from (3.6), which is of degree  $d(\mathbf{T}_L) + d(\Gamma') + d(\mathbf{T}_R)$ . As such, the degrees of  $\alpha(v)$ ,  $\beta(v)$  and  $\gamma(v)$  can be at most  $d(\Gamma') + d(\mathbf{T}_R)$ ,  $d(\mathbf{T}_L) + d(\mathbf{T}_R)$  and  $d(\mathbf{T}_L) + d(\Gamma')$ , respectively<sup>†</sup>. For a formal proof we refer to [LH89; Pet02].

To conclude, we mention that there are several workarounds to either satisfy or avoid the VEC. Moreover, there are related problems, where the starting point is not a curve network, but instead a quad-triangle mesh, possibly with provided tangent planes at the vertices [Pet90a]. Modification of the input data was already mentioned as a possible approach to satisfy the VEC. Another popular method is to split a single patch into multiple smaller patches, thereby creating more DoFs; work using these so-called *macro-patches* includes [Far83; Pip87; Pet90b; HB00; HBC08]. Less common is the approach to use a singular parameterisation [Pet91a]. Finally, using rational patches such as Gregory patches, the VEC can be avoided altogether [CK83; SS90; Man+92; Bos+12]. We briefly discuss this option in the next section.

### 3.1.2 Interpolating curve networks with Gregory patches

Let us go back to the approach of using bicubic patches to interpolate a given cubic curve network, using linear  $\alpha(v)$  and  $\beta(v)$  and a quadratic  $\lambda(v)$ . However, this time we *disregard* the cyclic nature of the problem. In other words, we only focus on the shared curves  $\Gamma(v)$  between two patches, and compute  $2 \times 2$  interior points for each  $\Gamma(v)$ . In that case, we have 5 vectorial equations, the first and last of which are automatically satisfied by the tangents of the curve network. We are then left with 3 equations to determine the 4 interior points. Combining these equations, we obtain the following system:

$$\begin{pmatrix} \alpha_0 & \beta_0 = 1 & 0 & 0 \\ \alpha_1 = 1 & \beta_1 & \alpha_0 & \beta_0 = 1 \\ 0 & 0 & \alpha_1 = 1 & \beta_1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_{2,1}^L \\ \mathbf{P}_{1,1}^R \\ \mathbf{P}_{2,2}^L \\ \mathbf{P}_{1,2}^R \end{pmatrix} = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \mathbf{H}_3 \end{pmatrix}, \quad (3.24)$$

with shorthand notation  $\mathbf{H}_k$  for the right-hand sides for  $k \in \{1, 2, 3\}$ . The same remarks as for (3.22) apply<sup>‡</sup>. Note that the  $\mathbf{H}_k$  contain the unknown coefficient  $\gamma_1$ ; its value can either be optimised for (with the objective function based on what follows below), or be obtained through degree-elevation from a linear  $\gamma(v)$ :

$$(1-v)\gamma_0 + v\gamma_1 = (1-v)^2\gamma_0 + 2(1-v)v \left[ \frac{1}{2}\gamma_0 + \frac{1}{2}\gamma_1 \right] + v^2\gamma_1.$$

Assuming the matrix in (3.24) is regular, the underdetermined system can be solved using a *minimum norm* approach with respect to some preferred solution [BP93]. For an interior patch, this results in 8 interior control points — two per corner. By rationally

<sup>†</sup>Note that (3.6) results in a set of *scalar* equations that are, in general, independent [DeR90], whereas (3.7) results in a set of *vectorial* equations.

<sup>‡</sup>And again, the matrix can be obtained directly from Figure 3.3 (top).

blending these pairs of control points [CK83], we obtain 4 interior control points;

$$\begin{aligned} P_{12} &= \frac{uP_{12}^v + (1-v)P_{12}^u}{u + (1-v)}, & P_{22} &= \frac{(1-u)P_{22}^v + (1-v)P_{22}^u}{(1-u) + (1-v)}, \\ P_{11} &= \frac{uP_{11}^v + vP_{11}^u}{u + v}, & P_{21} &= \frac{(1-u)P_{21}^v + vP_{21}^u}{(1-u) + v}, \end{aligned} \quad (3.25)$$

see also Figure 3.5. These blends can be associated with the bicubic Bernsteins  $\mathcal{B}_{kl}(u, v)$ , for  $k, l \in \{1, 2\}$ , and used to define a bicubic *Gregory patch*. The patch boundaries are still Bézier curves, though the interior of the patch is now rational rather than polynomial (which – in the interior – results in more complex partial derivatives). This is the price we pay for avoiding the VEC (the mixed second-order partial derivatives, or twist vectors, are no longer compatible) while still interpolating the network with patches connecting  $G^1$ .

A triangular Gregory patch can be defined along the same lines. Consider using quartic triangular patches to interpolate a curve network *without* sharing twist vectors between neighbouring patches. Following the same approach as above, we end up with pairs of control points at the corners once more (see Figure 3.5). There are various ways to blend them; Walton and Meek [WM96] proposed the following blend,

$$\begin{aligned} P_{211} &= \frac{vP_{211}^w + wP_{211}^v}{v + w}, \\ P_{121} &= \frac{wP_{121}^u + uP_{121}^w}{w + u}, \\ P_{112} &= \frac{uP_{112}^v + vP_{112}^u}{u + v}, \end{aligned} \quad (3.26)$$

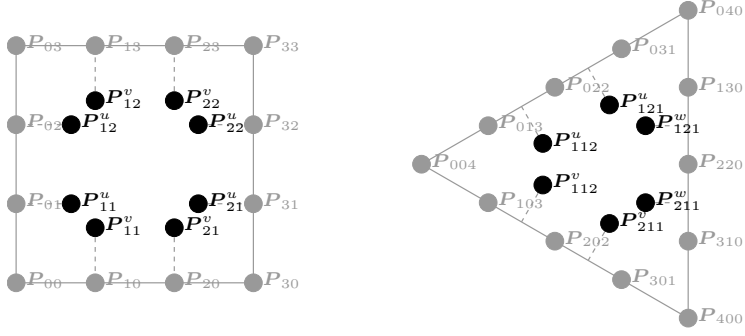
whereas Longhi [Lon85] proposed

$$\begin{aligned} P_{211} &= \frac{(1-v)wP_{211}^v + v(1-w)P_{211}^w}{(1-v)w + v(1-w)}, \\ P_{121} &= \frac{(1-w)uP_{121}^w + w(1-u)P_{121}^u}{(1-w)u + w(1-u)}, \\ P_{112} &= \frac{(1-u)vP_{112}^u + u(1-v)P_{112}^v}{(1-u)v + u(1-v)}. \end{aligned} \quad (3.27)$$

Observe that also for triangular Gregory patches, the patch boundaries are still Bézier curves.

In addition to introducing (3.25)<sup>†</sup>, [CK83] also introduced an interesting approach to achieve a  $G^1$  connection between two bicubic patches. Given the control points defining the boundary curves of a left- and a right patch meeting at  $\Gamma(v)$ , the left patch is temporarily replaced by a *basis patch*. This is a *virtual* bicubic patch with a quadratic transversal derivative. Regarding (3.7), if we then assume both  $\alpha(v)$  and  $\gamma(v)$  to be

<sup>†</sup>These actually follow as an application of *Gregory's square*, which was introduced by Gregory in the context of Coons patches [Gre74].



**Figure 3.5:** Bicubic Gregory patch (left) and quartic triangular Gregory patch (right).

linear and  $\beta(v)$  to be *constant*, we obtain

$$\alpha(v)T_L(v) + \gamma(v)\Gamma'(v) = T_R(v),$$

$$((1-v)\alpha_0 + v\alpha_1) \sum_{k=0}^2 b'_k \mathcal{B}_k^2(v) + ((1-v)\gamma_0 + v\gamma_1) \sum_{k=0}^2 c_k \mathcal{B}_k^2(v) = \sum_{k=0}^3 a_k \mathcal{B}_k^3(v),$$

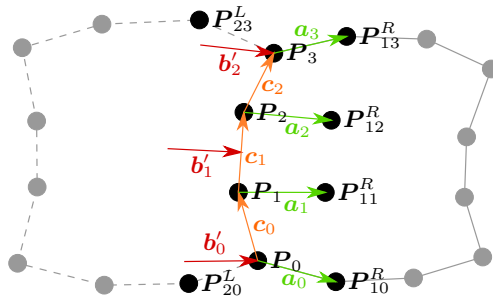
which uses notation reminiscent of that in the original paper (see Figure 3.6). Re-writing this, we then obtain the following expression,

$$\sum_{k=0}^3 \left( \frac{3-k}{3} \alpha_0 b'_k + \frac{k}{3} \alpha_1 b'_{k-1} + \frac{3-k}{3} \gamma_0 c_k + \frac{k}{3} \gamma_1 c_{k-1} - a_k \right) \mathcal{B}_k^3(v) = 0,$$

which results in the four equations

$$\alpha_0 b'_0 + \gamma_0 c_0 = a_0, \quad \frac{2}{3} \alpha_0 b'_1 + \frac{1}{3} b'_0 + \frac{2}{3} \gamma_0 c_1 + \frac{1}{3} \gamma_1 c_0 = a_1,$$

$$\frac{1}{3} \alpha_0 b'_2 + \frac{2}{3} b'_1 + \frac{1}{3} \gamma_0 c_2 + \frac{2}{3} \gamma_1 c_1 = a_2, \quad \alpha_1 b'_2 + \gamma_1 c_2 = a_3.$$



**Figure 3.6:** The  $G^1$  connection between a bicubic virtual basis patch with quadratic transversal derivative (left) and a bicubic patch (right).

With  $\mathbf{b}'_0$  and  $\mathbf{b}'_2$  defined as normalised versions of  $\mathbf{P}_{10}^R - \mathbf{P}_{20}^L$  and  $\mathbf{P}_{13}^R - \mathbf{P}_{23}^L$ , respectively<sup>†</sup>, the coefficients of  $\alpha(v)$  and  $\gamma(v)$  follow from the first and last equation. Then, with  $\mathbf{b}'_1 = \frac{\mathbf{b}'_0 + \mathbf{b}'_2}{2}$ , our unknowns  $\mathbf{P}_{11}^R$  and  $\mathbf{P}_{12}^R$  follow from the second and third equation.

If we apply the same approach by temporarily replacing the right patch by a basis patch, the  $\mathbf{b}'_k$  now opposite in direction, observe that the two basis patches then connect with  $C^1$  continuity. This means that, by transitivity<sup>‡</sup>, our original bicubic patches connect  $G^1$ .

The approach can be generalised to include quartic triangular patches. Applying it to all shared curves  $\Gamma(v)$  and subsequently blending the resulting pairs of interior control points at corners ultimately results in a composite Gregory surface that is overall  $G^1$ . In general, the result is visually pleasing [CK83; Chi86], but can still be further tuned as demonstrated in [SS90; FH12].

## 3.2 Subdivision surfaces

The first publications on bivariate subdivision appeared 40 years ago [DS78; CC78]. These works introduced spline-based subdivision schemes generalising the subdivision of uniform biquadratic and bicubic tensor product B-spline surfaces to surfaces of arbitrary manifold topology, as well as preliminary techniques to analyse the behaviour of the resulting surface. Since then, a multitude of other schemes has been proposed, such as schemes based on three- and four-directional box splines [Loo87; PR97; VZ01; Dod+09], and schemes that are *not* spline based, including interpolatory schemes [DLG90] and the  $\sqrt{3}$  scheme [Kob00]. For a more complete overview, we refer the reader to one of the surveys [Ma05; Cas12] or books [WW01; PR08; AS10] on subdivision surfaces.

Our discussion of subdivision surfaces starts with a quick study of subdivision curves [Sab10]. Recall that in Section 2.3.2 we introduced the two-scale relation. For uniform B-splines of low degree, it can be nicely illustrated through the projection of shifted and dilated hypercubes, though for higher degrees, the Fourier approach (see Section 2.3.2) is more practical. Indeed, obtaining the coefficients for a uniform cubic B-spline requires the projection of a unit tesseract divided into  $2^4 = 16$  smaller tesseracts, which is a bit challenging to visualise properly. In contrast, using the Fourier approach the coefficients are readily obtained. In fact, observe from (2.30) that the coefficients follow from the binomial theorem (and therefore correspond to a row of Pascal's triangle). As such, the coefficients can also be obtained by *discrete convolution* of  $[1, 1]$  and the coefficients of the B-spline of one degree lower, in this case,  $[1, 3, 3, 1]$ . The result is  $[1, 1] * [1, 3, 3, 1] = [1, 4, 6, 4, 1]$ , scaled by  $(\frac{1}{2})^{4 - \dim(\text{supp}(\mathcal{M}^3))} = \frac{1}{8}$ ; see (2.31).

Now, observe what happens for a uniform cubic B-spline curve  $C(t)$  upon applying

<sup>†</sup>The original paper defined them as unit vectors orthogonal to the shared boundary  $\Gamma(v)$ , but were updated to the current definition in a later paper [Chi86].

<sup>‡</sup>With the basis patches connecting  $C^1$  to each other, and individually  $G^1$  to our original cubic patches,  $G^1$  connectivity of our cubic patches follows.

the two-scale relation to the basis functions:

$$\begin{aligned}
 C(t) &= \sum_{k=0}^l \mathcal{M}^3(t-k) \mathbf{P}_k = \frac{1}{8} \sum_{k=0}^l \left( \mathcal{M}^3(2(t-k)) + 4\mathcal{M}^3(2(t-k-\frac{1}{2})) + \right. \\
 &\quad \left. 6\mathcal{M}^3(2(t-k-1)) + 4\mathcal{M}^3(2(t-k-\frac{3}{2})) + \mathcal{M}^3(2(t-k-2)) \right) \mathbf{P}_k \\
 &= \frac{1}{8} \left( \mathcal{M}^3(2t), \dots, \mathcal{M}^3(2(t-l-2)) \right) \begin{pmatrix} 1 & & & & & & & & & & \\ 4 & & & & & & & & & & \\ & 6 & 1 & & & & & & & & \\ & 4 & 4 & & & & & & & & \\ & 1 & 6 & 1 & & & & & & & \\ & & 4 & 4 & & & & & & & \\ & & 1 & 6 & \ddots & & & & & & \\ & & & 4 & & \ddots & & & & & \\ & & & 1 & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & & & & & 4 & & & \\ & & & & & & & 6 & & & \\ & & & & & & & 4 & & & \\ & & & & & & & 1 & & & \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \vdots \\ \mathbf{P}_l \end{pmatrix}. \quad (3.28)
 \end{aligned}$$

The refinement coefficients appear as shifted *columns* in the matrix above, and show that neighbouring  $\mathcal{M}^3(t-k)$  share  $d=3$  dilated and shifted versions of themselves. The more important observation, however, is that the *rows* of the matrix represent affine combinations<sup>†</sup>. Recall that we have seen something similar in Section 2.3.1 on knot-insertion. In fact, the approach is equivalent to global midpoint knot-insertion. It follows that the affine combinations can be applied to the control points  $\mathbf{P}_k$  in order to obtain the refined control polygon of the curve  $C(t)$ .

This concept is known as uniform B-spline subdivision. In this context, the sequence of coefficients appearing as columns is referred to as the subdivision *mask*. The affine combinations appearing as rows are the subdivision *stencils*. Note that the (unnormalised) stencils can be obtained from the mask by extracting every other coefficient from it.

Repeated application of the stencils to the control polygon results in ever closer approximations of the actual curve. Eventually, the control polygon converges to the curve itself. Isolating a suitable  $3 \times 3$  part of the subdivision matrix above allows for a quick demonstration:

$$\lim_{m \rightarrow \infty} \left( \frac{1}{8} \begin{pmatrix} 4 & 4 & \\ 1 & 6 & 1 \\ & 4 & 4 \end{pmatrix} \right)^m \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix} = \frac{1}{6} \begin{pmatrix} 1 & 4 & 1 \\ 1 & 4 & 1 \\ 1 & 4 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \end{pmatrix}, \quad (3.29)$$

which computes the *limit position* of  $\mathbf{P}_1$ , with  $[1, 4, 1]$  the *limit stencil*<sup>‡</sup>. Note that the coefficients of the limit stencil match the values of  $\mathcal{M}^3(t)$  evaluated at its interior knots,

<sup>†</sup>With the exception of the first- and last  $d=3$  rows.

<sup>‡</sup>Because stencils are always affine combinations, the normalising factor (in this case  $\frac{1}{6}$ ) is often omitted, and instead implied.

which also follows from its Bernstein-Bézier form (see Figure 2.19). This proves that the limit point lies on the curve itself; an alternative way to see this is to apply blossoming.

Bivariate subdivision generalises the notions briefly touched upon above — the two-scale relation resulting in the subdivision mask, the subdivision stencils, and most importantly, limit behaviour and analysis — to surfaces of arbitrary topology. We are mostly interested in the limit surface rather than its approximation by a control net that is repeatedly subdivided. Nevertheless, the latter is certainly useful in various settings, including the use of computer graphics in e.g. feature films as visual effects (VFX), animated movies or video games.

We focus on two different subdivision schemes to explain some of the theory behind subdivision surfaces. The first one, based on half-box splines, is the topic of the publication associated with this section [BSK19] and is discussed in detail. The second one is the well-known Catmull-Clark scheme that has been studied extensively since its publication over forty years ago [CC78]. We will be considerably more concise in this case and return to selected applications in Chapter 4.

### 3.2.1 Half-box spline subdivision

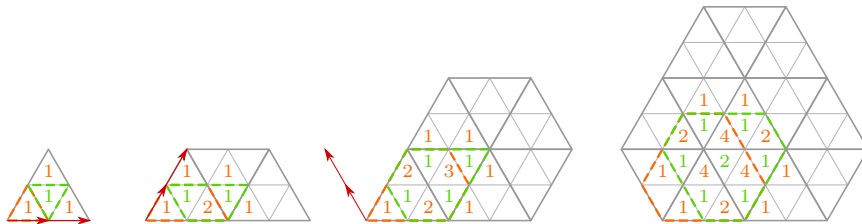
I first came across half-box splines while working on a literature review on box splines for my MSc thesis [Bar13]. Initially I thought them an oddity, but later realised that subdivision of a three-valent control net (composed of mostly hexagons) would be a nice addition to the available arsenal of bivariate subdivision schemes. In Cambridge I worked on the topic for a while together with Malcolm Sabin and Jiří Kosinka. After some initial tuning and encountering *ineffective eigenvalues* (both to be discussed below), I summarised our efforts and presented them at a workshop in Bernried, Germany (which was, coincidentally, my first). Then my focus shifted to another topic and I put the project on the ever-expanding virtual shelf of future projects. After gathering virtual dust for a couple of years, I returned to it — and here we are.

Although most work on bivariate subdivision focuses on triangular or quadrilateral meshes (or a combination thereof), a number of publications considers subdivision of three-valent meshes either as their main topic or as an illustrative example. As mentioned in [CBV02], hexagonal meshes can be subdivided using various factors, including 3, 4 and 7. Naturally, higher factors are also possible. However, the higher the factor, the faster the number of  $n$ -gons grows upon subdivision, which therefore makes it less attractive. Like most *honeycomb schemes*, [CBV02] construct a scheme that generates 3 new hexagons (or  $n$ -gons, in general) for each original one, and can be interpreted as a dual of the  $\sqrt{3}$  scheme. The standard subdivision tools are used to tune the stencils and verify the necessary properties of the characteristic map. Boundary rules are added in [BCV02]. Likewise, [AS02] propose a factor-3 honeycomb scheme using the same topological approach but with different stencils. The related paper [Akl+04] considers the subdivision of three-valent pentagonal meshes.

The above schemes are not spline-based, but instead have subdivision stencil tuning as starting point. In contrast, a spline-based scheme starts with the subdivision of ordinary regions of the piecewise polynomial surface and extends from there. We note that stencils for subdividing the ordinary regions of half-box spline surfaces are known [PB02]. Extraordinary regions can be either approached using a hole-filling approach

Interestingly, [OS03] briefly describe an approach on how to handle extraordinary regions in a half-box spline scheme using a composition of subdivision operations in one of its examples, but does not go into detail. [Dod05] also mentions the theoretical existence of a half-box spline scheme in a systematic overview of bivariate subdivision schemes. Finally, [DLL92] and in particular [DLS03] both consider factor-4 honeycomb schemes. These are however not spline based, although they use the same topological split.

We take off from the definition of half-box splines given in (2.23) with the aim to find the two-scale relation for  $H_{111}(u)$ , with  $u \in \mathbb{R}^2$ . With the freshly introduced notion of discrete convolution in mind, we consider the indicator function with upward-pointing triangle  $\triangle$  as support and quadrisect it by bisecting its edges. Note that this is the equivalent of bisecting the constant uniform B-spline  $\mathcal{M}^0(t)$  in the univariate setting. Subsequently, we apply directional convolution in each of the three directions  $e_1, e_2$  and  $e_3$ , stressing that also the direction vectors are bisected. Using this iterative approach, we obtain the coefficients of the two-scale relation for  $H_{111}(u)$ , or in subdivision terminology, its mask. Figure 3.7 illustrates the procedure.



Now, given a half-box spline surface

with  $H_{111}^k(u)$  shorthand notation for either a shifted copy of  $H_{111}(u)$  or a shifted copy of the mirror-symmetric  $\bar{H}_{111}(u)$ , we can follow the same approach as taken in (3.28) and express each  $H_{111}^k(u)$  as the sum of dilated, shifted, and in this case, also *rotated* copies of itself using the subdivision mask:

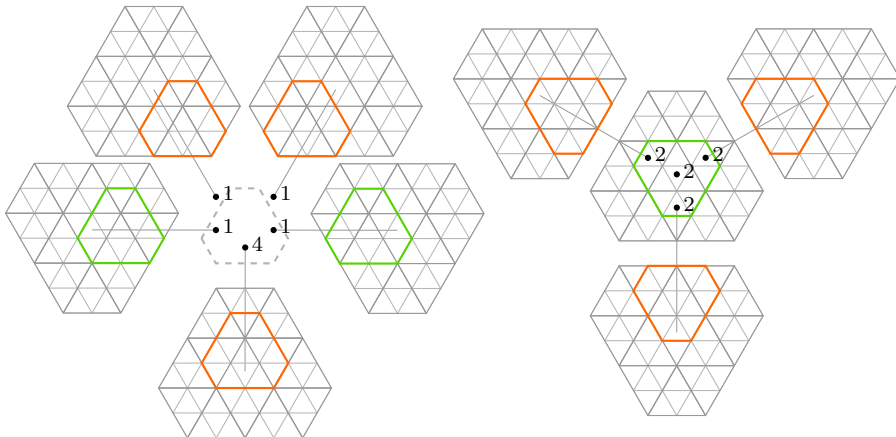
64

with  $m_l$  the mask coefficients. If, for a moment, we consider the so-called *functional setting* such that  $P_k \in \mathbb{R}$  with all  $P_k = 1$ , substituting (3.31) in (3.30) yields

$$\sum_k H_{111}^k(u) = \sum_k \sum_l m_l h_{111}^{kl}(u) = 1. \quad (3.32)$$

We can then rewrite the above such that each  $h_{111}^{kl}(u)$  appears only once<sup>†</sup>, multiplied by a sum of mask coefficients  $m_l$ . As the  $h_{111}^{kl}(u)$  sum to  $(\frac{1}{2})^{5-\dim(\text{supp}(H_{111}))} = (\frac{1}{2})^3 = \frac{1}{8}$ , this implies that the sums of coefficients are equal to 8. We then normalise so that the sums of coefficients are 1 (which in turn implies scaling  $h_{111}^{kl}$  by a factor 8). Going back to our spatial setting with  $P_k \in \mathbb{R}^3$ , this means that the sums of coefficients are affine combinations that are applied to the control points  $P_k$  (compare to (3.28)). In other words, these sums are our subdivision stencils.

The stencils can also be obtained following a more visual approach. Selecting a single  $h_{111}^{kl}(u)$ , we can determine all  $H_{111}^k(u)$  it contributes to. Writing down the relevant mask coefficient  $m_l$  next to the centre of each  $H_{111}^k(u)$  then results in a visualisation of the stencil; Figure 3.8 illustrates the approach.



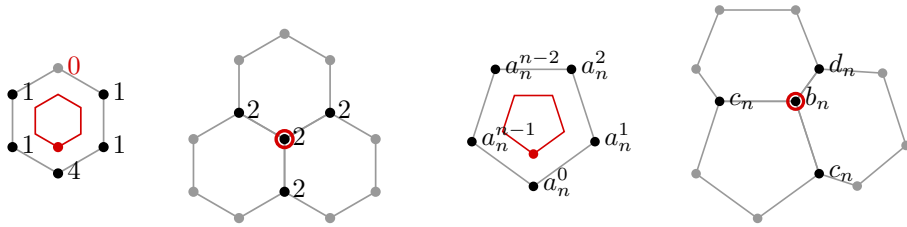
**Figure 3.8:** A visual approach to determine the two stencils associated with the half-box spline  $H_{111}(u)$ . The coloured outlines correspond to the mask coefficients shown in Figure 3.7.

With these two stencils,  $S_1$  and  $S_2$ , we can now subdivide a three-valent hexagonal control net (or *mesh*) and its corresponding half-box spline surface. However, as can easily be shown using the Euler-Poincaré characteristic, the set of surfaces that can be modelled using a three-valent hexagonal mesh is extremely limited in a topological sense. Considering closed meshes, these are surfaces that are homeomorphic to a torus. In order to model other surfaces, we have to mix in different  $n$ -gons (e.g. pentagons or heptagons), referred to as *extraordinary faces* (EFs), or allow vertices with valencies  $n \neq 3$ , so called *extraordinary vertices* (EVs). We limit ourselves to EFs<sup>‡</sup>, and therefore have to construct new stencils for general  $n$ -gons; see Figure 3.9.

<sup>†</sup> Similar to the example in the univariate case, each  $h_{111}^{kl}(u)$  contributes to multiple  $H_{111}^k(u)$ .

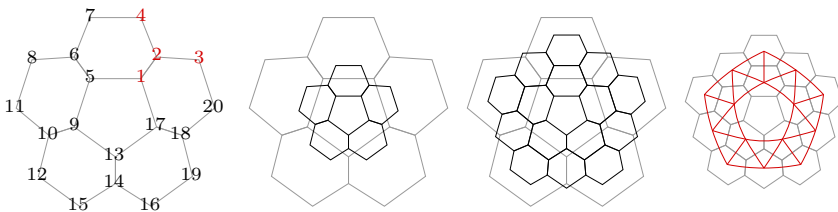
<sup>‡</sup> Allowing EVs in this scheme would correspond to allowing EFs in e.g. Loop's subdivision scheme, which generalises the subdivision of  $\mathcal{M}_{222}(u)$  to meshes containing EVs with valencies  $n \neq 6$ .





**Figure 3.9:** Half-box spline subdivision stencils  $S_1$  and  $S_2$  for a hexagonal mesh (left) and for a mesh containing general  $n$ -gons (right).

Determining the coefficients for the new stencils — such that the limit surface is visually pleasing — is the main challenge of developing a new subdivision scheme. A toolbox to facilitate this task has been developed and extended [DS78; BS86; BS88; BS90; Rei95; PR08] since the first subdivision schemes were published. In short, the following steps are taken. First, a mesh  $M$  is composed by surrounding an isolated  $n$ -gon by a ring of hexagons. Then, a general subdivision matrix  $\mathcal{A}_n$  that describes the subdivision of this mesh is established. Upon applying this matrix to the mesh, a special *spline ring* composed of spline patches (in our case, half-box spline patches) is constructed above the ordinary regions of the subdivided mesh (i.e. the parts not containing the EF). If this spline ring satisfies certain conditions, the limit surface is  $C^1$  at the limit position of the EF (i.e. its vertices have converged to a single point). Away from the EF, the limit surface is composed of the spline patches and has the usual continuity (which in our case is also  $C^1$ ). Figure 3.10 summarises the steps.



**Figure 3.10:** An isolated pentagon in an otherwise hexagonal mesh  $M$  and its vertex labelling (left). Subdivision of the mesh, resulting in two rings of hexagons surrounding a scaled pentagon (middle). Characteristic spline ring composed of half-box spline patches defined using the ordinary part of the subdivided mesh (right).

Using the notation for the stencil coefficients introduced in Figure 3.9 (right) and

the ordering of the vertices shown in Figure 3.10 (left), we construct  $\mathcal{A}_5$ :

$$\mathcal{A}_5 = \begin{pmatrix} \begin{array}{cccc|cccc|cccc|cccc} a_n^0 & 0 & 0 & 0 & a_n^1 & 0 & 0 & 0 & a_n^2 & 0 & 0 & 0 & a_n^3 & 0 & 0 & 0 & a_n^4 & 0 & 0 & 0 \\ b_n & d_n & 0 & 0 & c_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_n & 0 & 0 & 0 \\ 1/2 & 1/8 & 1/8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 1/8 & 0 & \textcolor{red}{0} \\ 1/2 & 1/8 & 0 & 1/8 & 1/8 & 1/8 & \textcolor{red}{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\ \hline \begin{array}{cccc|cccc|cccc|cccc} a_n^4 & 0 & 0 & 0 & a_n^0 & 0 & 0 & 0 & a_n^1 & 0 & 0 & 0 & a_n^2 & 0 & 0 & 0 & a_n^3 & 0 & 0 & 0 \\ c_n & 0 & 0 & 0 & b_n & d_n & 0 & 0 & c_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/8 & 1/8 & 0 & \textcolor{red}{0} & 1/2 & 1/8 & 1/8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/8 & 0 & 1/8 & 1/8 & 1/8 & \textcolor{red}{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \\ \hline \begin{array}{cccc|cccc|cccc|cccc} a_n^3 & 0 & 0 & 0 & a_n^4 & 0 & 0 & 0 & a_n^0 & 0 & 0 & 0 & a_n^1 & 0 & 0 & 0 & a_n^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_n & 0 & 0 & 0 & b_n & d_n & 0 & 0 & c_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/8 & 1/8 & 0 & \textcolor{red}{0} & 1/2 & 1/8 & 1/8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/8 & 0 & 1/8 & 1/8 & 1/8 & \textcolor{red}{0} & 0 & 0 & 0 & 0 & 0 \end{array} \\ \hline \begin{array}{cccc|cccc|cccc|cccc} a_n^2 & 0 & 0 & 0 & a_n^3 & 0 & 0 & 0 & a_n^4 & 0 & 0 & 0 & a_n^0 & 0 & 0 & 0 & a_n^1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_n & 0 & 0 & 0 & b_n & d_n & 0 & 0 & c_n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 1/8 & 0 & \textcolor{red}{0} & 1/2 & 1/8 & 1/8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/8 & 1/8 & 1/8 & 1/8 & \textcolor{red}{0} & 0 \end{array} \\ \hline \begin{array}{cccc|cccc|cccc|cccc} a_n^1 & 0 & 0 & 0 & a_n^2 & 0 & 0 & 0 & a_n^3 & 0 & 0 & 0 & a_n^4 & 0 & 0 & 0 & a_n^0 & 0 & 0 & 0 \\ c_n & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_n & 0 & 0 & 0 & b_n & d_n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/8 & 1/8 & 0 & \textcolor{red}{0} & 1/2 & 1/8 & 1/8 & 0 \\ 1/8 & 1/8 & 1/8 & \textcolor{red}{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/8 & 0 \end{array} \end{pmatrix}.$$

Observe that  $\mathcal{A}_5$  is a *block-circulant matrix* with  $4 \times 4$  lower-triangular blocks  $A_k$ . This is the case for all  $n \geq 3$ , defining  $\mathcal{A}_n$  as

$$\mathcal{A}_n = \begin{pmatrix} A_0 & A_1 & \dots & A_{n-1} \\ A_{n-1} & A_0 & \dots & A_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ A_1 & A_2 & \dots & A_0 \end{pmatrix}, \quad (3.33)$$

with the blocks  $A_k$  defined as

$$A_0 = \begin{pmatrix} a_n^0 & 0 & 0 & 0 \\ b_n & d_n & 0 & 0 \\ 1/2 & 1/8 & 1/8 & 0 \\ 1/2 & 1/8 & 0 & 1/8 \end{pmatrix}, \quad A_1 = \begin{pmatrix} a_n^1 & 0 & 0 & 0 \\ c_n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1/8 & 1/8 & \textcolor{red}{0} & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} a_n^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$\dots, \quad A_{n-2} = \begin{pmatrix} a_n^{n-2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_{n-1} = \begin{pmatrix} a_n^{n-1} & 0 & 0 & 0 \\ c_n & 0 & 0 & 0 \\ 1/8 & 1/8 & 0 & \textcolor{red}{0} \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The red zeroes in  $A_1$  and  $A_{n-1}$  come from the “skipped” vertex in stencil  $S_1$  (see Figure 3.9) but have no further significance.

Because  $\mathcal{A}_n$  is a square matrix, it can be applied arbitrarily many times to the mesh  $M$ , which causes it to converge to some limit mesh. The behaviour of the converging mesh is strongly related to the eigenstructure of the subdivision matrix. To study it, we apply the discrete Fourier transformation (DFT) to  $\mathcal{A}_n$ , which turns it into a *block diagonal* matrix  $\mathcal{D}_n$ . As we will see, the eigenstructure follows almost directly from  $\mathcal{D}_n$ . Before that, in order to gain some insight in the DFT, consider the definition of a point  $\omega$  on the unit circle in the complex plane:

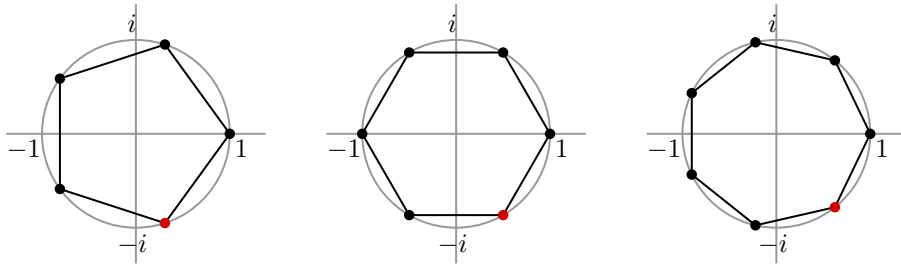
$$\omega = e^{-\frac{2\pi i}{n}} = \cos\left(\frac{2\pi}{n}\right) - i \sin\left(\frac{2\pi}{n}\right).$$

The DFT matrix  $F_n$ , with  $n$  the valency of the  $n$ -gon, is constructed such that each entry corresponds to the point  $\omega$  raised to the power  $kl$ , where the indices refer to the  $k^{th}$  row and  $l^{th}$  column of the matrix, both starting from 0. Note that this results in a symmetric matrix, and that  $\omega^{kl} = \omega^{kl-n}$ . The inverse of  $F_n$  is its conjugate transpose  $F_n^*$  divided by  $n$ . As example, consider  $n = 3$ :

$$F_3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega^4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^{-1} \\ 1 & \omega^{-1} & \omega \end{pmatrix},$$

$$F_3^{-1} = \frac{1}{3} F_3^*.$$

A geometric interpretation of multiplication by  $\omega$  is a clockwise rotation by  $\frac{2\pi}{n}$  on the unit circle. Each row (or column) of  $F_n$  is therefore associated with a *frequency*, that is, how fast the equally-spaced points on the complex unit circle are sampled. The zeroth row is constant, the first row goes around once, the second twice and so on. See Figure 3.11.



**Figure 3.11:** The point  $\omega$  on the unit circle for  $n \in \{5, 6, 7\}$  shown in red. Application of  $F_n$  results in clockwise rotations of the point. The resulting sequence of points (red and black) forms an  $n$ -gon inscribed in the unit circle.

In order to obtain  $\mathcal{D}_n$ , we first take the Kronecker product of the DFT matrix  $F_n$  with the unit matrix of the same size as the blocks  $A_k$ , which yields a block-DFT matrix  $\mathcal{F}_{n,4} = F_n \otimes I_4$ . We then obtain

$$\mathcal{D}_n = \mathcal{F}_{n,4}^{-1} \mathcal{A}_n \mathcal{F}_{n,4} = \begin{pmatrix} D_0 & & & \\ & D_1 & & \\ & & \ddots & \\ & & & D_{n-1} \end{pmatrix}. \quad (3.34)$$

The remarkable result is that the following expression captures all blocks  $D_l$ , with  $l \in [0, n-1]$  and for all valencies  $n \geq 3$ :

$$D_l = \sum_{k=0}^{n-1} \omega^{kl} A_k = \begin{pmatrix} \sum \omega^{kl} a_n^k & 0 & 0 & 0 \\ b_n + 2c_n \cos\left(\frac{2\pi l}{n}\right) & d_n & 0 & 0 \\ \frac{1}{2} + \omega^{-l} \frac{1}{8} & \frac{1}{8} + \omega^{-l} \frac{1}{8} & \frac{1}{8} & 0 \\ \frac{1}{2} + \omega^l \frac{1}{8} & \frac{1}{8} + \omega^l \frac{1}{8} & 0 & \frac{1}{8} \end{pmatrix}.$$

Observe that as all blocks  $A_k$  are *lower triangular*, the blocks  $D_l$  are as well. This means that the eigenvalues of  $\mathcal{D}_n$  follow directly as the union of diagonal entries of the blocks  $D_l$ . Regarding its *right* eigenvectors  $\vec{v}$ , we have

$$\begin{aligned}\mathcal{D}_n \vec{v} &= \lambda \vec{v} \\ \mathcal{F}_{n,4}^{-1} \mathcal{A}_n \mathcal{F}_{n,4} \vec{v} &= \lambda \vec{v} \\ \mathcal{A}_n \mathcal{F}_{n,4} \vec{v} &= \lambda \mathcal{F}_{n,4} \vec{v} \\ \mathcal{A}_n \vec{w} &= \lambda \vec{w},\end{aligned}\tag{3.35}$$

where  $\vec{w} = \mathcal{F}_{n,4} \vec{v}$ . It follows that the matrix  $\mathcal{D}_n$  is *similar* to  $\mathcal{A}_n$ . That is, they share the same *eigenvalues*, but in general have different *right eigenvectors*. The right eigenvectors of  $\mathcal{D}_n$  are the right eigenvectors of the blocks  $D_l$  vertically padded with zeroes. For the *left* eigenvectors  $\overleftarrow{v}$  of  $\mathcal{D}_n$  we can follow a similar approach and obtain  $\overleftarrow{w} = \overleftarrow{v} \mathcal{F}_{n,4}^{-1}$ , with  $\overleftarrow{w}$  the left eigenvectors of  $\mathcal{A}_n$ .

Subdivision surface theory [PR08] now states the following two well-known necessary conditions for a subdivision scheme to be  $C^1$ :

1.  $\mathcal{A}_n$  should have a single dominant eigenvalue of value 1 (which is satisfied by the top-left eigenvalue in  $D_0$  in our case),
2.  $\mathcal{A}_n$  should have two *subdominant* eigenvalues, coming from the blocks  $D_1$  and  $D_{n-1}$ .

These subdominant eigenvalues also indicate the *rate of contraction* of a subdivision scheme, which ideally should be  $\frac{1}{2}$  for a binary scheme [Don+16]. From the top-left entries of the  $D_l$  it now follows that

$$\lambda_l = \sum_{k=0}^{n-1} \omega^{kl} a_n^k, \text{ or } \boldsymbol{\lambda} = F_n \mathbf{a},\tag{3.36}$$

and therefore  $\mathbf{a} = F_n^{-1} \boldsymbol{\lambda}$ . Choosing a spectrum of  $\boldsymbol{\lambda} = [1, \frac{1}{2}, \frac{1}{4}, \dots, \frac{1}{4}, \frac{1}{2}]$  then gives

$$\begin{aligned}a_n^k &= \frac{1}{n} \sum_{l=0}^{n-1} \omega^{-kl} \lambda_j = \frac{1}{4n} \sum_{l=0}^{n-1} \omega^{-kl} + \frac{3}{4n} + \frac{1}{4n} (\omega^{-k} + \omega^k) \\ &= \frac{1}{4n} \sum_{l=0}^{n-1} \omega^{-kl} + \frac{3}{4n} + \frac{2}{4n} \cos\left(\frac{2\pi k}{n}\right),\end{aligned}$$

where all  $\lambda$  were replaced by  $\frac{1}{4}$  in the first step and corrected for  $l \in \{0, 1, n-1\}$  with missing parts  $[\frac{3}{4}, \frac{1}{4}, \frac{1}{4}]$ . We can further simplify this, as the  $\omega^{-kl}$  sum to  $n$  for  $k = 0$  and to 0 otherwise. As such, we obtain

$$a_n^k = \begin{cases} \frac{1}{4} + \frac{5}{4n} & k = 0, \\ \frac{3+2\cos(\frac{2\pi k}{n})}{4n} & \text{otherwise,} \end{cases}\tag{3.37}$$

which is in fact the same expression as the one which appeared in [DS78]. Note that for  $n = 6$  we have to use the original stencil  $S_1$ , as unfortunately it is not captured by (3.37). We will comment on this observation later on.

As for tuning stencil  $S_2$ , note that a modification of its coefficients would result in a subdivision scheme that is no longer *uniform*. In other words, a three-valent vertex would then be treated differently depending on the valency of the  $n$ -gon it is part of. For now we leave  $S_2$  to be unchanged, but we will come back to this towards the end of the section.

With the subdivision matrix  $\mathcal{A}_n$  complete, we can study what happens to the mesh  $M$  when it is subdivided arbitrarily many times. To do so, we consider the eigendecomposition  $\mathcal{A}_n = V\Lambda V^{-1}$ , where  $V$  contains the right eigenvectors of  $\mathcal{A}_n$  as columns,  $\Lambda$  is a diagonal matrix<sup>†</sup> containing the eigenvalues of  $\mathcal{A}_n$  and  $V^{-1}$  contains the left eigenvectors of  $\mathcal{A}_n$  as rows<sup>‡</sup>. Now, because  $\mathcal{A}_n$  contains a single dominant eigenvalue  $\lambda_0 = 1$ , observe that

$$\begin{aligned} \lim_{m \rightarrow \infty} \mathcal{A}_n^m M &= \lim_{m \rightarrow \infty} V \Lambda^m V^{-1} M \\ &= (\vec{w}_0 \ \vec{w}_1 \ \dots \ \vec{w}_{n-1}) \begin{pmatrix} \lambda_0 & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{pmatrix} \begin{pmatrix} \vec{w}_0^T \\ \vec{w}_1^T \\ \vdots \\ \vec{w}_{n-1}^T \end{pmatrix} M. \end{aligned} \quad (3.38)$$

Furthermore, as the individual rows of  $\mathcal{A}_n$  add up to 1, the right eigenvector  $\vec{w}_0$  associated with  $\lambda_0$  is a column of ones. It follows that (3.38) reduces to  $M$  (pre-)multiplied by the left eigenvector  $\vec{w}_0^T$  associated with  $\lambda_0$ , which shows that  $\vec{w}_0^T$  is in fact the *limit stencil* cf. (3.29).

We now arrive at the last step that comes with the third necessary condition for a scheme to be  $C^1$ :

3. The *characteristic map*, that is, the limit surface associated with the mesh defined by the right eigenvectors  $\vec{w}_1$  and  $\vec{w}_{n-1}$  associated with the subdominant eigenvalues  $\lambda_1$  and  $\lambda_{n-1}$ , should be injective and regular.

Let us pause here to see why  $\vec{w}_1$  and  $\vec{w}_{n-1}$  can be interpreted as the  $x$ - and  $y$ -coordinates of vertices forming a mesh. First, note that the blocks  $D_1$  and  $D_{n-1}$  are complex conjugates, which means that also their eigenvectors are complex conjugates. Determining the  $(4 \times 1)$  eigenvector  $\vec{g}$  of  $D_1$  associated with  $\lambda_1 = \frac{1}{2}$  and vertically padding it with zeroes gives us  $\vec{v}_1$ . Using the DFT, in particular the first column  $[1, \omega, \omega^2, \dots, \omega^{(n-1)}]^T$  of  $F_n$ , then results in  $\vec{w}_1$ ; see (3.35). Observe that  $\vec{w}_1$  contains  $n$  counter-clockwise rotations of  $\vec{g}$ . Similarly, we obtain  $\vec{v}_{n-1}$  from  $D_{n-1}$  (though note that  $\vec{v}_1$  and  $\vec{v}_{n-1}$  are *not* complex conjugates). Applying the DFT matrix again, more

<sup>†</sup>A subdivision matrix can be *defective*, in which case it cannot be diagonalised. Instead, the Jordan normal form can then be used, resulting in generalised eigenvectors and a matrix  $\Lambda$  that is *block*-diagonal.

<sup>‡</sup>Recall that the dot product of right- and left eigenvectors associated with *different* eigenvalues vanishes. The dot product of right- and left eigenvectors associated with the *same* eigenvalue can be assumed to be 1 by scaling the eigenvectors. It follows that  $V^{-1}$  contains the left eigenvectors.

specifically, the last column  $[1, \omega^{-1}, \omega^{-2}, \dots, \omega^{-(n-1)}]^T$  of  $F_n$ , yields  $\vec{w}_{n-1}$ , containing  $n$  clockwise rotations of  $\vec{g}^*$ , the complex conjugate of  $\vec{g}$ . It follows that also  $\vec{w}_1$  and  $\vec{w}_{n-1}$  are complex conjugates. Visualising either in the complex plane  $\mathbb{C}$  shows a (logically) rotationally symmetric structure that can be interpreted as a mesh, which is known as the *natural configuration* [BS86]. Note that  $\vec{w}_1$  and  $\vec{w}_{n-1}$  can be redefined as its real and imaginary components, respectively. In that case, interpreting them as  $x$ - and  $y$ -coordinates of the vertices naturally yields the same result in  $\mathbb{R}^2$ .

Next, we express the mesh  $M$  in terms of the right eigenvectors, that is, find coefficients  $c_k \in \mathbb{R}^3$  such that

$$V \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = M.$$

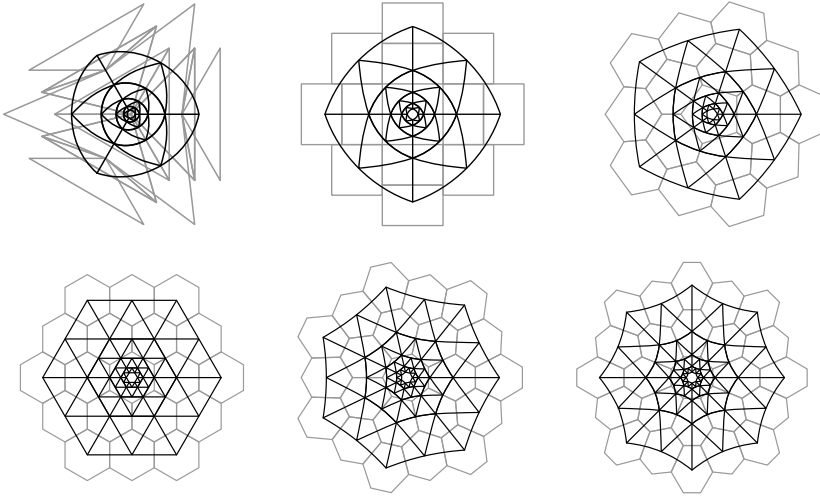
This can always be done as the  $\vec{w}_k$  form a basis. The coefficients follow as  $V^{-1}M$ . Then, we have that

$$\begin{aligned} \mathcal{A}_n^k M &= V \Lambda^k V^{-1} V (V^{-1} M) \\ &= (\lambda_0^k \vec{w}_0 \quad \lambda_1^k \vec{w}_1 \quad \dots \quad \lambda_{n-1}^k \vec{w}_{n-1}) \begin{pmatrix} c_0 = \overleftarrow{w}_0^T M \\ c_1 = \overleftarrow{w}_1^T M \\ \vdots \\ c_{n-1} = \overleftarrow{w}_{n-1}^T M \end{pmatrix}. \end{aligned} \quad (3.39)$$

If we now assume the mesh  $M$  to be translated such that its limit position coincides with the origin, i.e.  $c_0 = \overleftarrow{w}_0^T M = \mathbf{0}$ , the subdominant components  $c_1 \vec{w}_1 + c_{n-1} \vec{w}_{n-1}$  become dominant upon repeatedly subdividing  $M$ . In other words, *any* mesh locally converges to (a linear transformation of) the natural configuration. The associated limit surface is referred to as the characteristic map, which – as stated above – should be injective and regular for the scheme to be  $C^1$ . Effectively, this means that we can locally use the characteristic map as parameter domain of the limit surface, and furthermore, that the Jacobian of the characteristic map does not vanish.

In addition, the above suggests that the plane containing the natural configuration is tangent to the limit surface at the limit position of the EF. This tangent plane is spanned by the two coefficients (i.e. vectors)  $c_1$  and  $c_{n-1}$ , which reveals that  $\overleftarrow{w}_1$  and  $\overleftarrow{w}_{n-1}$  are *tangent limit stencils* (note that the coefficients of each  $\overleftarrow{w}_m$  sum to zero for  $m \geq 1$ ). With  $c_1 = (1, 0, 0)$  and  $c_{n-1} = (0, 1, 0)$  we retrieve the natural configuration.

Observe that with increasing steps of subdivision, an increasingly large part of the mesh becomes ordinary. As shown in Figure 3.10 (right), these ordinary regions then can be associated with spline rings. When this is applied to the natural configuration, we obtain *characteristic spline rings*. Because of rotational symmetry and the scaling property of these spline rings composing the characteristic map (recall that the natural configuration is an eigenvector), analysis of the Jacobian on one sector of a single ring suffices to verify that the third necessary condition is met [WW01]. Figure 3.12 shows characteristic maps for selected valencies when using the original  $S_2$ .



**Figure 3.12:** Natural configurations (grey) and images of the characteristic map for  $n \in \{3, 4, 5, 6, 7, 8\}$  (black).

Although the characteristic map satisfies the required conditions for all depicted valencies, the natural configurations for  $n = 3$  and  $n = 4$  exhibit unusual behaviour. In both cases there are coinciding vertices, resulting in degenerate polygons. Furthermore, for  $n = 3$  the natural configuration contains self-intersecting polygons. For applications where the subdivided mesh (as opposed to the limit surface) is the object of interest, this could lead to undesirable results. As such, we can reconsider tuning  $S_2$  to potentially improve things.

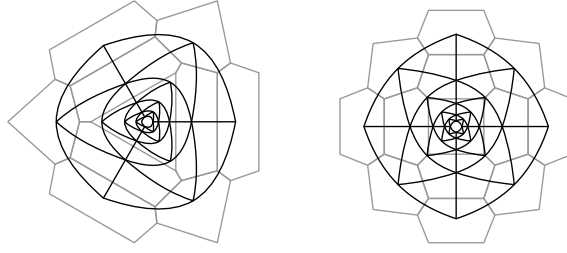
When tuning  $S_2$ , the condition for bounded curvature at the limit position of the  $n$ -gons comes into play, which states that

4.  $\mathcal{A}_n$  should have three *subsubdominant* eigenvalues  $\mu$ , equal to the squared value of the subdominant eigenvalues, coming from the blocks  $D_0$ ,  $D_2$  and  $D_{n-2}$ .

In our case, we have to ensure that  $\mu = \frac{1}{4}$ . The only way to satisfy this is to set  $d_n = \mu$ . Clearly, we also have the condition  $b_n + 2c_n + d_n = 1$ , which now yields the condition  $b_n + 2c_n = \frac{3}{4}$ . This leaves us with only one DoF, which then theoretically results in a range of admissible values of  $b_n$  and  $c_n$ . Figure 3.13 shows improved natural configurations and their associated characteristic maps using empirical values of  $b_3 = 0.45$  and  $b_4 = 0.35$ .

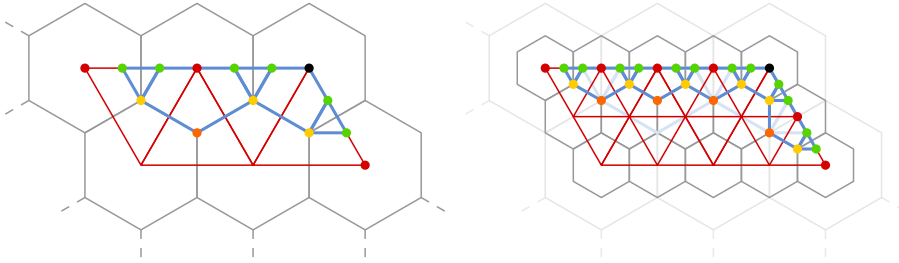
### Boundaries and creases

Support for boundaries in a subdivision scheme, and likewise for (sharp) creases, significantly increases the versatility of the objects that can be modelled with it. Adding boundary rules to honeycomb schemes has been considered in [BCV02], and for selected three-directional box splines in [SB03]. In our case, we have cubic surface patches with  $C^1$  connectivity, which hints at the use of cubic univariate subdivision with double knots [KSD13].



**Figure 3.13:** Improved natural configurations and their characteristic maps for  $n = 3$  (left) and  $n = 4$  (right).

Figure 3.14 illustrates our approach at an ordinary boundary (i.e. a contiguous set of hexagons at the boundary of a three-valent mesh). The Bézier points controlling the boundary curve are highlighted in red, green and black. Using e.g. blossoming [Ram89], it is readily observed that the green points form the control polygon of a cubic B-spline curve with double knots. The subdivision scheme for such a curve, which can act as either a boundary or as a sharp crease, is known [KSD13]. Corners of the mesh (such as the black point) can be reflected in the B-spline curve by using a triple knot.



**Figure 3.14:** Original (left) and subdivided (right) hexagonal mesh (grey) with triangular surface patches (red). The red, green and black points are the Bézier points defining the boundary curves (which is piecewise  $C^1$  cubic). At the boundary, the hexagons are truncated into pentagons and triangles to form an auxiliary structure (blue).

We use the control polygon of the B-spline to truncate the hexagons at the boundary. Rather than using half-polygons as used in [BCV02], we build an auxiliary structure composed of pentagons and triangles (blue), which is defined by the green and black Bézier boundary points and the interior points (yellow and orange). Showing this structure to the user instead of the contiguous set of hexagons at the boundary clearly provides a more intuitive preview of the eventual boundary curve associated with the mesh.

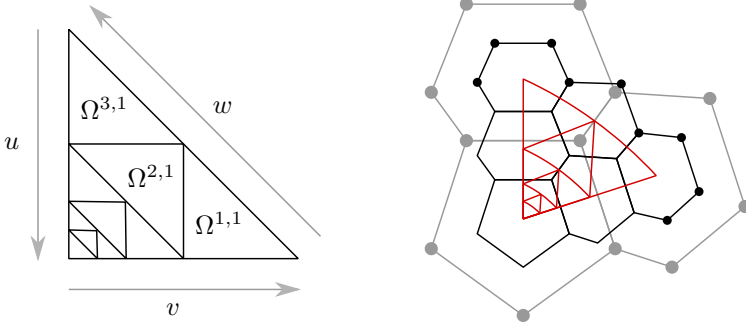
Direct interaction with the boundary control polygon proves difficult. Indeed, moving a single green point requires the update of other points of the boundary hexagon it lies within, which in turn influence the position of neighbouring green points. As such, interaction quickly turns into solving a global system, which is not an acceptable approach. We defer handling interaction, along with handling the various types of corners (there are two convex and two concave configurations) to future research.



## Evaluation of the limit surface

With the subdivision stencils tuned and the  $C^1$  smoothness requirements satisfied, it is time to take a more pragmatic look at our scheme. Given a three-valent mesh of arbitrary manifold topology, possibly with boundaries, we are interested in its associated limit surface (in the context of our half-box spline subdivision scheme) and the evaluation thereof. As mentioned before, ordinary regions correspond to half-box spline surface patches. The remaining regions either contain EFs or lie on the boundary. Focusing on the former case, we assume that all EFs are separated in the sense that no two EFs share a common edge. In case EFs are not separated, the mesh can be (locally) subdivided by applying the appropriate stencils.

The extraordinary regions now form control nets  $N$  for the remaining surface patches. These control nets are composed of the  $n$ -gon (i.e. the EF) and two hexagons, and therefore consist of  $n + 7$  control points. We assume all patches to be parameterised on the unit triangle  $\Omega$ . As illustrated for the characteristic map before, every subdivision step effectively changes part of the extraordinary region into an extension of the ordinary region, which then becomes associated with ordinary surface patches. As such, the resulting extraordinary patch  $S(u)$  is composed of an infinite sequence of layers of three ordinary patches each. The unit triangle is partitioned such that each ordinary patch  $S(u)|_{\Omega^{k,l}}$  corresponds to a tile  $\Omega^{k,l}$ , where  $k \in \{1, 2, 3\}$  indexes the tiles in the layer  $l$ . Figure 3.15 shows an overview.



**Figure 3.15:** The domain triangle  $\Omega$  partitioned into tiles  $\Omega^{k,l}$  (left). A control net  $N$  containing an EF (grey) and its associated surface patch with the partitioning of the parameter domain projected onto it (right). The control net subdivided using  $\bar{\mathcal{A}}_n^S$  is shown in black.

We can now extend Stam’s approach for evaluating limit surfaces of other subdivision schemes [Sta98a; Sta98b] to our scheme. Given barycentric coordinates  $(u, v, w)$  in the domain triangle, we can readily determine the indices of the tile  $\Omega^{k,l}$  containing the point – the layer can be obtained as  $l = \text{floor}(-\log_2(u)) + 1$ , whereas the value for  $k$  then follows from  $v$  and/or  $w$ . To virtually subdivide the control net  $N^\dagger$ , we construct  $(n + 7) \times (n + 7)$  subdivision matrices  $\mathcal{A}_n^S$ . We apply this matrix  $(l - 1)$  times to  $N$ . In addition, we need  $(n + 16) \times (n + 7)$  extended subdivision matrices  $\bar{\mathcal{A}}_n^S$  to obtain the 9 vertices in the ordinary region, see Figure 3.15 (right). We apply this matrix once.

<sup>†</sup>Virtual subdivision refers to local subdivision of a region of interest in memory, not affecting the actual mesh.

Using  $13 \times (n + 16)$  extraction matrices  $X_n^k$  we can then extract the appropriate 13 control points associated with our ordinary patch. Finally, using maps  $t_{k,l}(u, v, w)$  we map  $\Omega^{k,l}$  to  $\Omega$  to evaluate the patch.

A key point of Stam's approach is to eigendecompose the subdivision matrix  $\mathcal{A}_n^S = V_n \Lambda_n V_n^{-1}$ , which potentially speeds up the computation of powers of  $\mathcal{A}_n^S$ . Summarising, we have

$$S(u) \Big|_{\Omega^{k,l}} = \mathbf{H}_{111}^T(t_{k,l}(u)) X_n^k \bar{\mathcal{A}}_n^S (V_n \Lambda_n^{l-1} V_n^{-1}) N, \quad (3.40)$$

with  $\mathbf{H}_{111}^T(u)$  the blending functions, i.e. the 13 parts of the half-box spline  $H_{111}(u)$ . The eigendecomposition also reveals two alternative representations of (3.40) in terms of either a set of *subdivision splines*  $\varphi_{n,m}(u)$  or a set of *eigenfunctions*  $\psi_{n,m}(u)$  that are defined as

$$\varphi_{n,m}(u) \Big|_{\Omega^{k,l}} = \mathbf{H}_{111}^T(t_{k,l}(u)) X_n^k \bar{\mathcal{A}}_n^S V_n \Lambda_n^{l-1} z_{n,m}, \quad (3.41)$$

with  $z_{n,m}$  the  $m^{\text{th}}$  column of  $V_n^{-1}$ , and

$$\psi_{n,m}(u) \Big|_{\Omega^{k,l}} = \lambda_{n,m}^{l-1} \mathbf{H}_{111}^T(t_{k,l}(u)) X_n^k \bar{\mathcal{A}}_n^S \vec{w}_{n,m}, \quad (3.42)$$

with  $\lambda_{n,m}$  and  $\vec{w}_{n,m}$  the  $m^{\text{th}}$  eigenvalue and right eigenvector of  $\mathcal{A}_n^S$ , respectively. This allows us to express the patch  $S(u)$  as

$$S(u) = \sum_{m=0}^{n+7} \varphi_{n,m}(u) \mathbf{P}_m = \sum_{m=0}^{n+7} \psi_{n,m}(u) c_m, \quad (3.43)$$

where  $\mathbf{P}_m$  are the control points forming the mesh  $N$ , and  $c_m$  are the coefficients when  $N$  is expressed in terms of the right eigenvectors, see (3.39). In addition, observe that the eigenfunctions satisfy the scaling relation<sup>†</sup>

$$\psi_{n,m}(u/2) = \lambda_{n,m} \psi_{n,m}(u). \quad (3.44)$$

From (3.42) it follows that  $\psi_{n,0}(u) = 1^{\ddagger}$ . Moreover, the scaling relation (3.44) hints at the following (though does not directly prove it) – the eigenfunctions associated with  $\lambda_{n,m} = \frac{1}{2}$  are linear, and those associated with  $\mu_{n,m} = \frac{1}{4}$  are *piecewise* quadratic.

Taking (directional) derivatives of the eigenfunctions results (due to the definition of  $t_k(u)$  and the chain rule) in a multiplicative factor  $2^l$ . Together with the already present  $\lambda_{n,m}^{l-1}$ , this yields a factor  $2(2\lambda_{n,m})^{l-1}$ . As the derivative of  $\psi_{n,0}(u)$  vanishes, this shows that the eigenfunctions associated with the subdominant eigenvalue  $\lambda_{n,m} = \frac{1}{2}$  dominate the behaviour of the limit surface near the limit point. In fact, these eigenfunctions *are* the characteristic map.

Finally, we mention that the evaluation of limit surfaces near boundaries is a straightforward extension, which follows the methods described in [SES04; LB07], both based on so-called *ghost points* and the Jordan normal form of subdivision matrices.

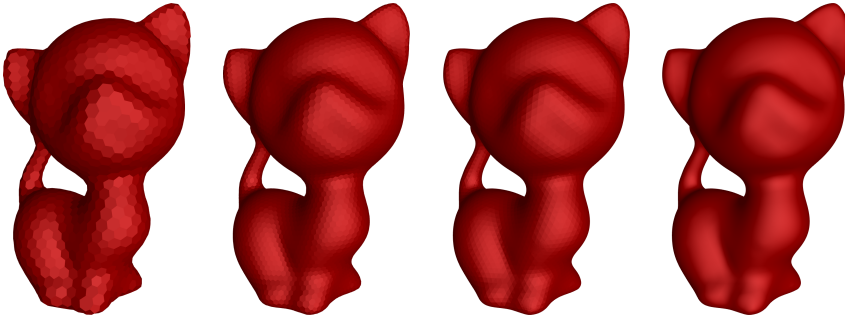
<sup>†</sup>If the point  $u$  is contained in tile  $\Omega^{k,l}$ ,  $u/2$  is contained in  $\Omega^{k,l+1}$ . Then,  $\mathbf{H}_{111}^T(t_{k,l}(u)) = \mathbf{H}_{111}^T(t_{k,l+1}(u/2))$ , and the relation follows.

<sup>‡</sup>Recall that  $\vec{w}_{n,0}$  is a vector of ones, that the rows of any subdivision matrix sum to 1, and that the parts of  $H_{111}(u)$  form a partition of unity. Then, it must be that  $\psi_{n,0}(u) = \lambda_0 = 1$ .

To recap this section — we can now evaluate limit surfaces associated with our scheme at arbitrary points; Figure 3.16 shows an example of a half-box spline subdivision surface evaluated at and around a pentagon, and Figure 3.17 the subdivision of a mesh I like to refer to as *Hexakitten*. In addition, we have gained insight in the representation of the surface patches using either subdivision splines or eigenfunctions.



**Figure 3.16:** A half-box spline subdivision surface evaluated at and around a pentagon. The resulting spline rings are offset for visualisation purposes only. The result is flat-shaded on the left to highlight the individual patches, and smooth-shaded on the right.



**Figure 3.17:** Initial mesh of Hexakitten (left), one step of half-box spline subdivision (middle left), the flat-shaded limit surface (middle right) and the smooth-shaded limit surface (right).

### Ineffective eigenvectors

To conclude our discussion on half-box spline subdivision, we discuss the notion of *ineffective eigenvectors* [PR08]. Given a set of blending functions that is *not* linearly independent, a nullspace (kernel) associated with these blending functions exists. Observe that this is the case for our  $\mathbf{H}_{111}^T(u)$ , which contains 13 blending functions (i.e. the parts of  $H_{111}(u)$ ) to represent a cubic patch. However, *at most* 10 basis functions are required to represent a cubic patch. Computation (using the BB-form) shows that the associated nullspace of our blending functions is of dimension 3.

Next, consider the evaluation of an extraordinary patch  $S(u)$  as in (3.43) using the eigenfunctions  $\psi_{n,m}(u)$  of (3.42). Note that  $\bar{\mathcal{A}}_n^S$  extends the right eigenvector  $\vec{w}_{n,m}$  of

length  $(n+7)$  to a vector  $\vec{w}_{n,m}$  of length  $(n+16)$ , and that  $X_n^k$  subsequently extracts 13 of its entries, resulting in a vector  $\vec{w}_{n,m}$ . Now, if  $\vec{w}_{n,m} \in \ker \left( \mathbf{H}_{111}^T(u) \right)$ , it follows that  $\psi_{n,m}(u)$  vanishes on the tiles  $\Omega^{k,l}$  on all layers for this specific  $k$ . In fact, it turns out that  $\psi_{n,m}(u)$  vanishes on the entire domain  $\Omega$ . If the associated eigenvalue  $\lambda_{n,m} = 0$ , this clearly has no effect, but if  $\lambda_{n,m} \neq 0$ , it does. In the latter case, the eigenvector  $\vec{w}_{n,m}$  is referred to as *ineffective*.

The consequence of an ineffective eigenvector is that part of the spectrum of  $\mathcal{A}_n^S$  does not contribute to the resulting surface. This can pose a problem for the analysis of a subdivision scheme if eigenvectors associated with the dominant, subdominant or subsubdominant eigenvalues coming from the appropriate block(s)  $D_l$  are ineffective. Ineffective eigenvectors associated with other eigenvalues do not affect the analysis and can be considered harmless.

A possible approach is to check for each individual eigenvector whether it is effective or not, and if it is not, to remove it by modifying the subdivision matrix [PR08]. Alternatively, it is sufficient to verify whether the set of eigenvectors relevant for  $C^1$  continuity (and possibly those associated with bounded curvature) is effective; if one or more turn out to be ineffective, they can be removed using the same procedure.

For the ordinary valency  $n = 6$ , the vectors  $\vec{w}_{n,m}$  contain the same entries as the original eigenvectors  $\vec{w}_{n,m}$  (though in different order). As such, we can directly check whether they are in the kernel of our blending functions. It turns out that in this case there are three ineffective eigenvectors. To our knowledge, this is the first non-artificial subdivision scheme that exhibits this phenomenon. The three ineffective eigenvectors are associated with eigenvalues  $\frac{1}{2}, \frac{1}{4}, \frac{1}{4}$ , which looks problematic at first sight. However, eigenanalysis in the Fourier domain shows that these eigenvalues do not come from relevant blocks  $D_l$ . These ineffective eigenvectors are therefore harmless, although it results in the peculiar property that our half-box spline scheme has a *triple* subdominant eigenvalue.

### 3.2.2 Catmull–Clark subdivision

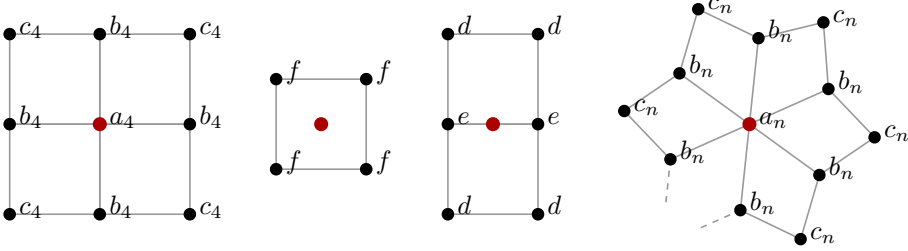
The Catmull–Clark subdivision scheme is a generalization of midpoint knot-refinement for uniform bicubic B-spline surfaces. The associated mask can easily be obtained by taking the tensor-product of the mask for uniform cubic B-splines (see Section 3.2):

$$\begin{array}{|c|} \hline 1 \\ \hline 4 \\ \hline 6 \\ \hline 4 \\ \hline 1 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline c_4 & d & b_4 & d & c_4 \\ \hline d & f & e & f & d \\ \hline b_4 & e & a_4 & e & b_4 \\ \hline d & f & e & f & d \\ \hline c_4 & d & b_4 & d & c_4 \\ \hline \end{array}.$$

The stencils can then be extracted from the mask by selecting every other entry from every other row. They are highlighted in different shades and are visualised in Figure 3.18.

The tensor-product stencils suffice for the subdivision of quad meshes for with all vertices have valency  $n = 4$ , though just as with three-valent meshes, we need to allow EVs or EFs in the mesh to permit the design of meshes of arbitrary manifold topology.

Catmull–Clark supports both, though after the first subdivision step, all  $n$ -gons are turned into quadrilaterals. As such, we focus on stencils to update EVs. Developing these can be done in various ways. The original paper by Catmull and Clark [CC78] used an empirical approach, whereas [DS78] used a more analytical method to derive coefficients  $a_n$ ,  $b_n$  and  $c_n$  for the extraordinary stencil (see Figure 3.18, right).



**Figure 3.18:** Stencils for the Catmull–Clark subdivision scheme. The ordinary vertex stencil (left) is generalised to arbitrary valencies  $n \geq 3$  (right).

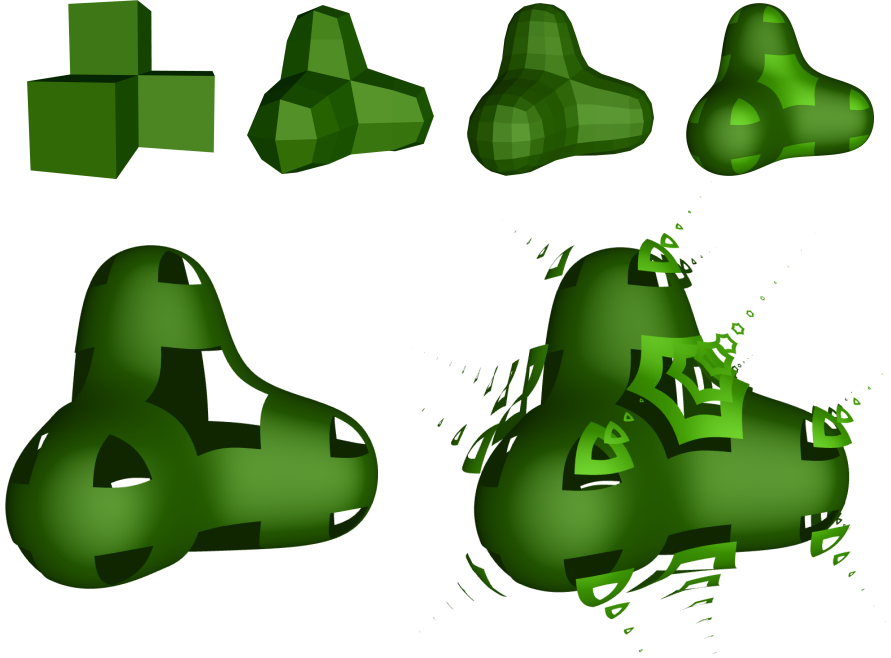
Although various improvements have been suggested (see e.g. [ADS06; ZSC18] and the references therein), the rules proposed by Catmull and Clark are still the most popular ones. These stencil coefficients can be expressed as simple expressions valid for every valence  $n \geq 3$ :

$$a_n = 1 - \frac{7}{4n}, \quad b_n = \frac{3}{2n^2}, \quad c_n = \frac{1}{4n^2}. \quad (3.45)$$

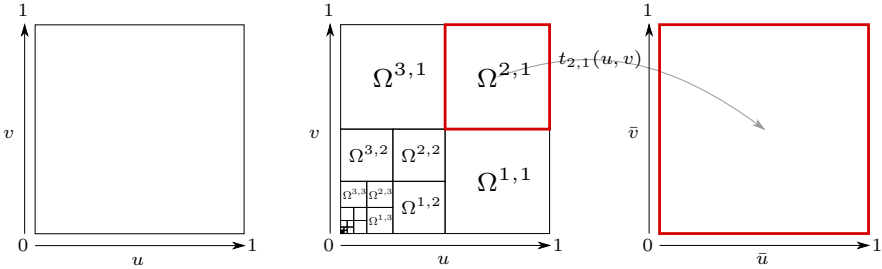
We now have a complete set of stencils and can apply Catmull–Clark subdivision to any closed quadrilateral mesh. Adding boundary rules to this scheme is straightforward, as the boundaries are simply uniform cubic B-splines. However, it should be mentioned that – especially for concave sections of the boundary – the default boundary stencils can be improved [BLZ00]. Figure 3.19 shows a mesh containing several EVs of valencies  $n \in \{3, 5, 6\}$ , two subdivision steps and the resulting limit surface.

Although incomplete attempts at proving  $C^1$  continuity for Catmull–Clark appeared not long after the original publication, a complete proof was published years later in [PR98] based on the characteristic map. We note that for  $n \neq 4$ , the subdominant eigenvalues  $\lambda \neq \frac{1}{2}$ , which leads to different rates of contraction around EVs. Furthermore, for  $n \neq 4$  the subsubdominant values  $\mu \neq \lambda^2$ , which results in zero or unbounded curvature at the limit point. Modifications have been proposed to ameliorate this [ADS06]. The scheme does not exhibit ineffective eigenvectors as the uniform bicubic B-splines are linearly independent.

Evaluation of arbitrary points on the limit surface can be done by Stam’s method, whose original publication [Sta98a] focused solely on the Catmull–Clark scheme. Every quadrilateral surface patch is parameterised on the unit square  $\Omega$ , which is partitioned into tiles  $\Omega^{k,l}$ , see Figure 3.20. The patch consists of an infinite number of layers of three tiles each, all connecting with  $C^2$  continuity. Only at the EV does the continuity decrease to  $C^1$ . For a given point  $(u, v)$ , the layer is determined as  $\text{floor}(-\log_2(\max(u, v))) + 1$ , after which  $k$  follows using  $u$  and/or  $v$ .



**Figure 3.19:** An initial mesh followed by two steps of Catmull–Clark subdivision and the corresponding limit surface (top). The ordinary regions of the limit surface can be directly evaluated as uniform bicubic B-splines (bottom left). The extraordinary regions are filled with spline rings connecting  $C^2$ , though the continuity is reduced to  $C^1$  at the limit position of the EV (bottom right).



**Figure 3.20:** Every quadrilateral surface patch is parameterised on the unit square  $\Omega$  (left), which is partitioned into tiles  $\Omega^{k,l}$  (middle). Before an ordinary patch associated with a tile  $\Omega^{k,l}$  can be evaluated, the tile needs to be mapped to  $\Omega$  using a map  $t_{k,l}(u, v)$  (right).

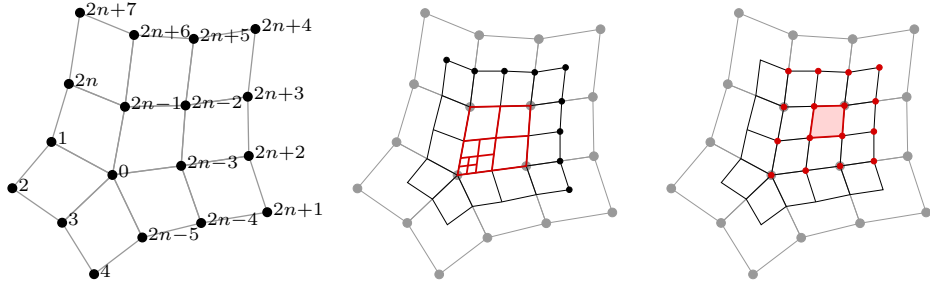
For each valency, a  $(2n+8) \times (2n+8)$  subdivision matrix  $\mathcal{A}_n^S$  is constructed, which is applied  $(l-1)$  times as virtual subdivision to the control net  $N$  for a patch  $S(u, v)$ . The matrix can be eigendecomposed as  $\mathcal{A}_n^S = V_n \Lambda_n V_n^{-1}$ . An extended subdivision matrix  $\tilde{\mathcal{A}}_n^S$  of size  $(2n+17) \times (2n+8)$  is applied afterwards. To extract the 16 control points required to evaluate the ordinary patch associated with the tile  $\Omega^{k,l}$ , we use extraction matrices  $X_n^k$  of size  $16 \times (2n+17)$ . Finally, the tile is mapped to  $\Omega$  using a

transformation  $t_{k,l}(u, v)$ . Altogether, we have

$$S(u, v) \Big|_{\Omega^{k,l}} = \mathbf{M}_{44}^T(t_{k,l}(u, v)) X_n^k \bar{\mathcal{A}}_n^S (V_n \Lambda_n^{l-1} V^{-1}) N, \quad (3.46)$$

with  $\mathbf{M}_{44}^T(u, v)$  the 16 parts of the uniform bicubic B-spline. Figures 3.20 and 3.21 illustrate the approach. Note that our ordering of control points differs from the one Stam uses — ours slightly simplifies the implementation of  $\mathcal{A}_n^S$ .

We remark that similarities in notation regarding the evaluation of half-box spline surfaces and Catmull–Clark surfaces are on purpose; from the context it is clear which scheme is intended.



**Figure 3.21:** The control net  $N$  for a quadrilateral face containing an EV with  $n = 5$  along with the ordering of its control points (left). The associated patch is composed of an infinite number of layers, each composed of three ordinary patches. Using an extended subdivision matrix, 9 additional control points are obtained (middle). The 16 control points for evaluating an ordinary patch are then extracted using  $X_n^k$  (right).

Expressions for the subdivision splines  $\varphi_{n,m}(u, v)$  and eigenfunctions  $\psi_{n,m}(u, v)$  cf. (3.41) and (3.42) follow from (3.46) as

$$\varphi_{n,m}(u, v) \Big|_{\Omega^{k,l}} = \mathbf{M}_{44}^T(t_{k,l}(u, v)) X_n^k \bar{\mathcal{A}}_n^S V_n \Lambda_n^{l-1} z_{n,m}, \quad (3.47)$$

and

$$\psi_{n,m}(u, v) \Big|_{\Omega^{k,l}} = \lambda_{n,m}^{l-1} \mathbf{M}_{44}^T(t_{k,l}(u, v)) X_n^k \bar{\mathcal{A}}_n^S \vec{w}_{n,m}. \quad (3.48)$$

Catmull–Clark subdivision is without a doubt the most commonly used subdivision scheme in the entertainment industry. Software packages such as MAYA and BLENDER include efficient implementations of the scheme. More recently, Pixar open-sourced its set of libraries collectively referred to as OPENSUBDIV using e.g. the GPU to allow high-performance rendering of the limit surface (or an approximation thereof).

The presence of subdivision modelling is less prominent in CAD/CAE software packages, though recently some have adopted this workflow. Notable examples include the introduction of the *Realize Shape* environment in NX 9, the *Mesh Modeler* in AutoCAD 2010, the *Imagine & Shape* module in CATIA 5 and the *Freestyle* extension in PTC CREO.

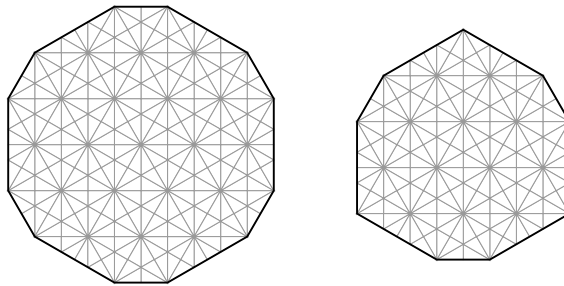
In Chapter 4 we will get back to Catmull–Clark subdivision splines and derive improved quadrature rules for efficient numerical integration.

### 3.2.3 An overview of (half-)box spline-based subdivision schemes

With any (half-)box spline satisfying the two-scale relation, the list of potential subdivision schemes is virtually endless. However, the majority of these schemes would not be very practical – associated surface patches of high degree or a lack of (rotational) symmetry of the spline quickly renders a scheme useless. As such, a list of relatively low degree (half-)box splines that are ‘mostly’ symmetric provides a decent overview of the possibilities in the context of subdivision schemes. Table 3.1 aims to show exactly this information. For the masks and stencils for most of these splines, we refer to [Dod+09].





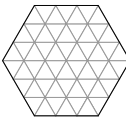

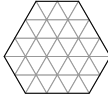


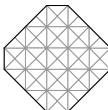
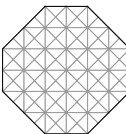
Note that some numbers in the fifth column have been highlighted. Green indicates that the patch is *undercomplete*, i.e. that the blending functions do not span the complete approximation space. In the case of  $M_{222}$ , the patch is quartic, which is typically defined using 15 basis functions. However, the box spline only supplies 12 blending functions, which means that 3 of them are missing (which is indicated with the  $-3$  in brackets). On the other hand, red indicates that a patch is *overcomplete*, i.e. that there are too many blending functions (indicating linear dependence). The surplus of blending functions is indicated in brackets. For this last group of blending functions, the occurrence of ineffective eigenvectors is likely, but not certain. Take for instance the Zwart–Powell spline  $M_{1111}$ , using 7 blending functions to define a quadratic patch. Eigenanalysis shows that, although there is an eigenvector that is in the kernel of these blending functions, its associated eigenvalue is 0, which means it is not classified as an ineffective eigenvector.

Secondly, there are a couple of remarks to be made with regard to half-box splines. These are only shown for the three-directional setting (note that in the first column the associated box spline is mentioned in grey). Although half-box splines can also be defined for the two- and four-directional cases, the resulting supports of the splines do not have a nice rotational symmetry. However, there is another group of box splines – the six-directional ones – for which there are splines with rotational symmetry similar to half-box splines. Figure 3.22 shows  $M_{111111}$  and  $H_{000111}$ , where the fourth, fifth and sixth directions are secondary directions each resulting from summing two principal directions.



**Figure 3.22:** The quartic  $C^3$  six-directional box spline  $M_{111111}$  (left) and a cubic  $C^2$  spline  $H_{000111}$  (left), obtained by applying directional convolution to the triangular indicator function in the fourth, fifth and sixth direction.



Spline	Support	Degree	Continuity	Blending functions per patch	Subdivision scheme
$M_{33}$		$2 \times 2$	$C^1$	$3 \times 3$	Doo-Sabin [DS78]
$M_{44}$		$3 \times 3$	$C^2$	$4 \times 4$	Catmull-Clark [CC78]
$M_{d+1d+1}$		$d \times d$	$C^{d-1}$	$(d+1) \times (d+1)$	Stam [Sta01]
$M_{222}$		4	$C^2$	12 (-3)	Loop [Loo87]
$M_{333}$		7	$C^4$	27 (-9)	—
$H_{111}$ ( $M_{221}$ )		3	$C^1$	7 + 6 (+3)	Ours [BSK19]
$H_{222}$ ( $M_{332}$ )		6	$C^3$	19 + 18 (+9)	—
$M_{1111}$		2	$C^1$	7 (+1)	Simplest [PR97]
$M_{2211}$		4	$C^2$	14 (-1)	Mentioned in [PS04b]
$M_{1122}$		4	$C^2$	17 (+2)	—
$M_{2222}$		6	$C^4$	28	4-8 [VZ01]

**Table 3.1:** An overview of the properties of mostly symmetric (half-)box splines of relatively low degree.

As mentioned above, not all subdivision schemes are based on (half-)box splines, and neither are all schemes uniform and stationary. An overview of the multitude of remaining schemes is somewhat out of scope. Notwithstanding, we briefly mention some special schemes, or rather, concepts. One of those is known as *guided subdivision*, which yields curvature-continuous surfaces by sampling piecewise polynomial

guide surfaces around EVs [KP07]. Another approach is referred to as *polar subdivision*, which doubles the valency of specific vertices every subdivision step and converges to a  $C^2$  surface [MP09]. Finally, there is the *NURBS-compatible subdivision scheme*, which extends subdivision surfaces to the realm of NURBS [Cas10].

Finally, we remark that there might be other extensions of subdivision that have not yet been explored in detail. One direction are the *rep-tiles*, which are (2D) shapes that can be composed of scaled, shifted and often rotated versions of themselves. Using a rep-tile as the support of an indicator function<sup>†</sup> and subsequently following the directional convolution approach would then guarantee a two-scale relation for the resulting function<sup>‡</sup>. Irrep-tiles expand this class considerably by allowing different scaling factors — an extreme example is the hexagon, which can be tessellated using an infinite number of smaller hexagons using scaling factors that are powers of  $\frac{1}{3}$ . The concepts can be extended to higher dimensions, for which I think *self-tessellating* shapes would be an apt description. See also [Pet14], which discusses related matters.

Proceeding into the same direction brings up the notion of *self-tiling tilesets* [Sal12; Sal14]. The idea would be to work with  $k$  functions that can be expressed as weighted sums of dilated and shifted versions of each other. Although in a different setting, an example is discussed in [LS07], which considers two (piecewise) bilinear functions, each with a square support, where one is rotated by  $45^\circ$  and scaled by a factor  $\sqrt{2}$ . It is placed in the context of four-directional box-splines, and are actually referred to as half-box splines, though it does not match the definition we have used so far (i.e. a pair of half-box splines should sum to a box spline of the same type). The example also shows some aspects of an irrep-tile, though note that the dilated versions of these functions have to overlap in order to compose each other — which is of course a fundamental aspect of a (generalised) two-scale relation, as overlap enables connections of higher continuity. Naturally, for both extensions, the shifts (and rotations) of the selected and resulting function(s) should still partition unity in order to be fit for subdivision.

## 3.3 Alternatives

In addition to composite  $G^1$  Bézier and Gregory surfaces and subdivision surfaces, there are a few other ways to obtain or model a smooth surface of arbitrary topology. We shine light on some of these, but do not elaborate.

### 3.3.1 Approximated Catmull–Clark subdivision surfaces

This approach is a combination of subdivision surfaces and composite  $G^1$  Bézier or Gregory surfaces. The idea is to replace the infinite sequences of spline rings at EVs (or EFs) by either Bézier patches [LS08] or by Gregory patches [Loo+09]. Arguably, there are very few applications where the exact shape of the subdivision surface around the

<sup>†</sup>It appears to be quite difficult to use something different from a constant function (i.e. the indicator function) such that the result still adheres to a two-scale relation.

<sup>‡</sup>Although the shape of some rep-tiles might not be practical to use for surface patches, the directions in which it is convolved might divide it into one or more shapes better suitable for the task (compare this to four-directional box-splines, where the initial square indicator function is split into  $45^\circ$ – $45^\circ$ – $90^\circ$  triangles upon directional convolution).

EV is the desired shape. As such, approximation is a very sensible approach. It can be interpreted as a method in between repeated subdivision of an initial mesh to get a visually smooth result (usually after 4-6 steps) and the actual exact limit surface.

The method using Gregory patches [Loo+09] also supports quad-triangle meshes (designers often add an occasional triangle in an otherwise quadrilateral mesh). It uses the quartic Gregory patch from Walton and Meek [WM96], though the work by Longhi [Lon85] is cited, which is not correct in their context.

The OPENSUBDIV library supports a very similar (though not identical) approach to approximate Catmull–Clark limit surfaces, and has recently added support for Loop subdivision surfaces as well.

### 3.3.2 T-spline surfaces

The original T-splines [Bak01] were developed to merge multiple NURBS surfaces defined using different knot vectors (resulting in T-sections in the mesh, hence the name T-splines). This is accomplished through the use of point-based splines (PB-splines), which are rational splines not constrained to a grid layout. In order to support surfaces of arbitrary manifold topology, they were combined with subdivision surfaces into T-NURCCS [Sed+03] using the NURSS framework [Sed+98]. More recent versions of T-splines appear to have replaced the subdivision approach around extraordinary vertices in the control net (referred to as *star points* in T-spline terminology) by  $G^1$  biquartic patches [Sco+13].

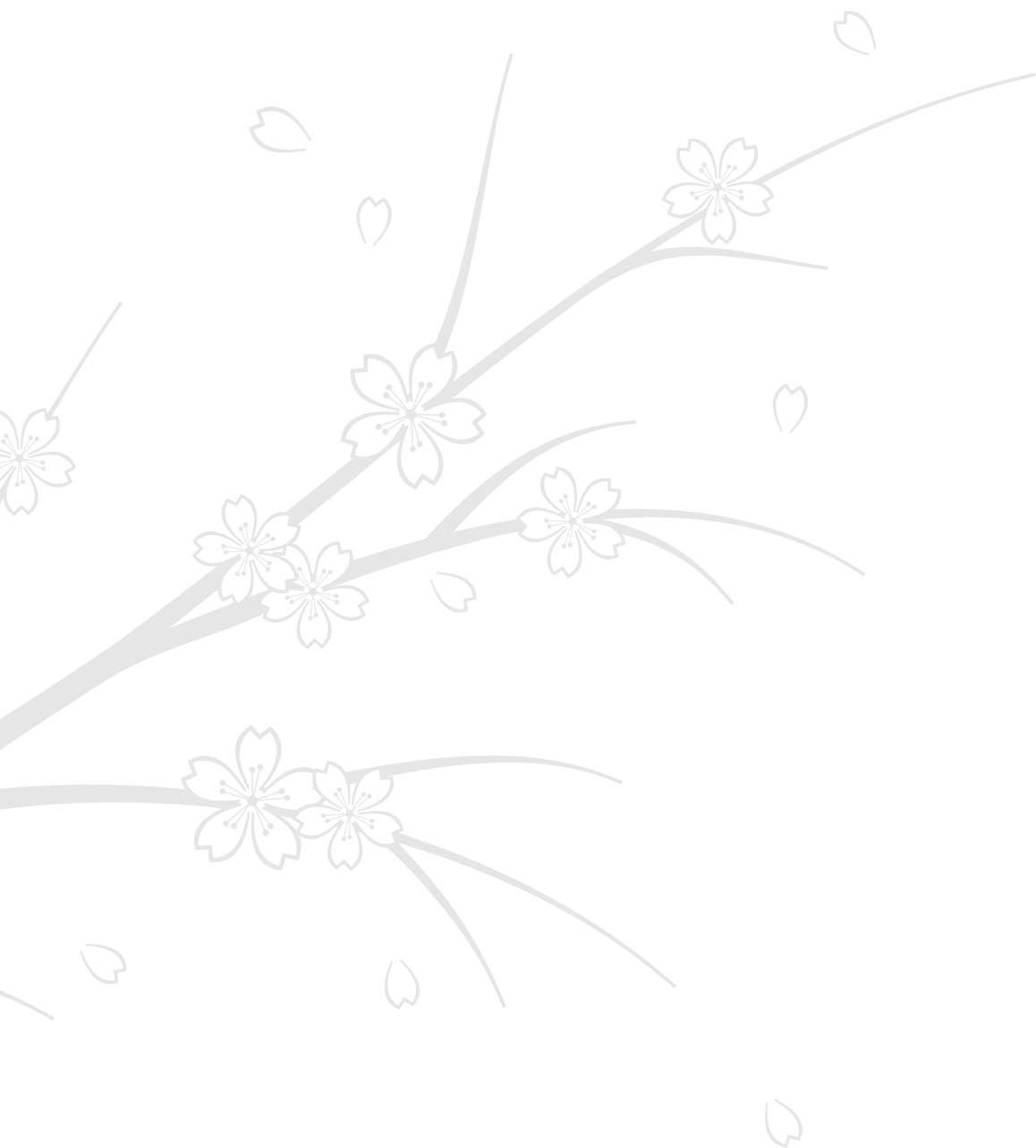


## 4 | Spline-based numerical methods



Parts of this chapter have been published as

- Pieter J Barendrecht, Michael Bartoň and Jiří Kosinka. “Efficient quadrature rules for subdivision surfaces in isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 340 (2018), pp. 1–23. See Section 4.2.2.
- Francesco Greco, Pieter J Barendrecht, Laurens Coox, Onur Atak and Wim Desmet. “Finite element analysis enhanced with subdivision surface boundary representations”. In: *Finite Elements in Analysis and Design* 137 (2017), pp. 56–72. See Section 4.4.1.



Numerical methods cover a vast area of research. In this chapter, we mostly restrict ourselves to numerical methods for solving partial differential equations (PDEs). The main focus lies with finite element analysis (FEA) and the closely related approach nowadays known as isogeometric analysis (IgA). In addition, we briefly discuss the boundary element method (BEM) and spline-enhanced FEM as alternatives.

## 4.1 The finite element method (FEM)

There are many different flavours of the finite element method, some specialised in certain types of PDEs, others borrowing aspects of related numerical methods to extend and improve. Here, we will only discuss and apply *linear* FEM [Joh12; Bat96; ZTZ05; Wei16; Hug87]. This still allows us to consider various PDEs on a variety of domains. Although the discussion of the univariate case is instructive – in addition to the functional setting, it can also be used to solve PDEs on curves embedded in  $\mathbb{R}^2$  or  $\mathbb{R}^3$  – we will focus on the bivariate case. The application of FEM on planar domains in  $\mathbb{R}^2$  allows us to introduce and explain the most relevant concepts. Similar to the univariate case, it can be applied to surfaces embedded in  $\mathbb{R}^3$  with only minor modifications (e.g. using the Laplace-Beltrami operator [Wal15] instead of the Laplace operator). It is also a popular approach to study PDEs on manifolds capturing the behaviour of (thin) shells following the descriptions of e.g. Kirchhoff-Love or Reissner-Mindlin [CB10]. Finally, it should be mentioned that FEM can be applied to volumes (solids), though in the context of e.g. isogeometric analysis [CHB09] there is still much work to be done in this area. This is the main reason for us to look into BEM [Bec92; SPG08] and spline-enhanced FEM [SFH08] towards the end of this chapter.

The running example in this section considers Poisson’s equation on a planar domain for an unknown scalar field  $u$ . We consecutively discuss the strong and weak formulations of the problem, the discretisation, numerical integration (quadrature) and (local) refinement.

### 4.1.1 Variational formulation

Given a region  $\Omega \in \mathbb{R}^2$  with its boundary  $\Gamma = \Gamma_D \cup \Gamma_N$  such that  $\Gamma_D \cap \Gamma_N = \emptyset$ , we define the following boundary value problem (BVP) for the Poisson equation:

$$-\Delta u = -\nabla \cdot \nabla u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f \quad \text{in } \Omega, \quad (4.1)$$

$$u = b_D \quad \text{on } \Gamma_D, \quad (4.2)$$

$$\frac{\partial u}{\partial n} = \nabla u \cdot n = b_N \quad \text{on } \Gamma_N, \quad (4.3)$$

with  $f$ ,  $b_D$  and  $b_N$  all known functions of  $(x, y)$ , and  $n$  the outward unit normal to the boundary  $\Gamma$ . Note that (4.2) and (4.3) are the Dirichlet and Neumann boundary conditions<sup>†</sup>, also referred to as *essential* and *natural* boundary conditions in this setting. The objective is to find the unknown  $u \in H^1(\Omega)$ , where  $H^1(\Omega)$  is the Hilbert space whose elements and their first order partial derivatives belong to  $L^2(\Omega)$ , i.e. are square integrable. Together, this is known as the *strong form*.

We now multiply both sides of (4.1) by an arbitrary *test function*  $v \in H_0^1(\Omega) = H^1(\Omega)$  such that  $v = 0$  on  $\Gamma_D$ . Subsequently, we integrate both sides over the domain,

<sup>†</sup>In the case of domain symmetry, it often suffices to consider only a part of the domain, though it requires the introduction of symmetric boundary conditions.

so we obtain

$$-\int_{\Omega} v \Delta u \, d\Omega = \int_{\Omega} v f \, d\Omega \quad \forall v \in H_0^1(\Omega). \quad (4.4)$$

Next, recall the divergence theorem

$$\int_{\Omega} \nabla \cdot A \, d\Omega = \int_{\Gamma} A \cdot n \, d\Gamma, \quad (4.5)$$

with  $A$  any vector-valued function on  $\Omega$ . If we set  $A = v \nabla u$ , we get (by applying the product rule)

$$\int_{\Omega} \nabla \cdot (v \nabla u) \, d\Omega = \int_{\Omega} \nabla v \cdot \nabla u + v \Delta u \, d\Omega = \int_{\Gamma} (v \nabla u) \cdot n \, d\Gamma = \int_{\Gamma} v \frac{\partial u}{\partial n} \, d\Gamma,$$

which is also known as Green's first identity. Rewriting (4.4) then yields

$$\begin{aligned} \int_{\Omega} \nabla v \cdot \nabla u \, d\Omega &= \int_{\Omega} v f \, d\Omega + \int_{\Gamma_D} v \frac{\partial u}{\partial n} \, d\Gamma + \int_{\Gamma_N} v \frac{\partial u}{\partial n} \, d\Gamma \\ &= \int_{\Omega} v f \, d\Omega + \int_{\Gamma_N} v b_N \, d\Gamma \quad \forall v \in H_0^1(\Omega), \end{aligned} \quad (4.6)$$

where the integral over  $\Gamma_D$  has vanished because  $v \in H_0^1(\Omega)$ . This is known as the *variational formulation* or *weak form*<sup>†</sup>; the requirements on the continuity of our unknown  $u$  are weaker compared to the strong form (4.1).

### 4.1.2 Discretisation, meshing and element types

Our next step is to discretise the weak form. As such, we need to select a subspace of  $H^1(\Omega)$ ; in FEA, this is typically some piecewise polynomial space  $V_h$ . Denoting the basis functions spanning our subspace as  $N_k(x, y)$ , we express a discretisation of our unknown  $u(x, y)$  as

$$u_h(x, y) = \sum_k u_k N_k(x, y) = \mathbf{u}^T \mathbf{N}(x, y) = \mathbf{N}^T(x, y) \mathbf{u}, \quad (4.7)$$

with  $u_k \in \mathbb{R}$ . Following the (Bubnov-)Galerkin approach, we express discretisations of any  $v$  in a similar way as

$$v_h(x, y) = \sum_k v_k N_k(x, y) = \mathbf{v}^T \mathbf{N}(x, y) = \mathbf{N}^T(x, y) \mathbf{v}. \quad (4.8)$$

Substituting both in (4.6), we obtain

$$\begin{aligned} \int_{\Omega} \nabla v_h \cdot \nabla u_h \, d\Omega &= \int_{\Omega} v_h f \, d\Omega + \int_{\Gamma_N} v_h b_N \, d\Gamma \quad \forall v_h \in V_h \\ &\Downarrow \\ \mathbf{v}^T \left( \int_{\Omega} \nabla \mathbf{N} (\nabla \mathbf{N})^T \, d\Omega \right) \mathbf{u} &= \mathbf{v}^T \int_{\Omega} \mathbf{N} f \, d\Omega + \mathbf{v}^T \int_{\Gamma_N} \mathbf{N} b_N \, d\Gamma \quad \forall \mathbf{v}^T, \end{aligned} \quad (4.9)$$

<sup>†</sup>The weak form can be expressed more compactly as  $a(v, u) = l_1(v, f) + l_2(v, b_N) \quad \forall v \in H_0^1(\Omega)$ , with  $a(v, u)$  a symmetric bilinear form and  $l_k(v, \cdot)$  linear forms. It is known as the *abstract form*.



where

$$(\nabla \mathbf{N})^T = \begin{pmatrix} \frac{\partial N_0(x,y)}{\partial x} & \frac{\partial N_1(x,y)}{\partial x} & \cdots & \frac{\partial N_m(x,y)}{\partial x} \\ \frac{\partial N_0(x,y)}{\partial y} & \frac{\partial N_1(x,y)}{\partial y} & \cdots & \frac{\partial N_m(x,y)}{\partial y} \end{pmatrix}.$$

As (4.9) has to hold for all  $\mathbf{v}^T$ , we can omit these columns of coefficients, which then results in

$$\mathbf{K} \mathbf{u} = \mathbf{q}, \quad (4.10)$$

with

$$\begin{aligned} \mathbf{K} &= \int_{\Omega} \nabla \mathbf{N} (\nabla \mathbf{N})^T d\Omega, \\ \mathbf{q} &= \int_{\Omega} \mathbf{N} f d\Omega + \int_{\Gamma_N} \mathbf{N} b_N d\Gamma. \end{aligned} \quad (4.11)$$

$\mathbf{K}$  is called the *stiffness matrix*. Note that the entry at  $(k, l)$  – both indices starting from zero – corresponds to  $\nabla N_k(x, y) \cdot \nabla N_l(x, y)$ , which shows that  $\mathbf{K}$  is symmetric<sup>†</sup>. We use  $\mathbf{q}$  as a general notation for the right-hand side of the system.

It is often convenient to express  $f$  and  $b_N$  in terms of the basis functions  $N_k$  as  $f = \sum_k f_k N_k = \mathbf{N}^T \mathbf{f}$  and  $b_N = \mathbf{N}^T \mathbf{b}_N$ , where in the latter case only the basis functions with support on the boundary are associated with (nonzero) coefficients. The resulting matrices  $\int_{\Omega} \mathbf{N} \mathbf{N}^T d\Omega$  and  $\int_{\Gamma} \mathbf{N} \mathbf{N}^T d\Gamma$  are known as the *mass matrix*  $\mathbf{M}$  and the *boundary mass matrix*  $\mathbf{B}$ , respectively.

Finally, we can incorporate the Dirichlet boundary condition by projecting the given function  $b_D$  onto the basis, which results in nodal values (i.e. coefficients  $\mathbf{b}_D$ )<sup>‡</sup>. The stiffness matrix can then be *condensed*, i.e. the rows of  $\mathbf{K}$  corresponding to the Dirichlet coefficients are removed and the associated columns are subsequently moved to the right-hand side. It might be useful to capture this procedure in a condensed example,

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} + \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}, \quad (4.12)$$

where the node associated with a Dirichlet boundary condition has been highlighted in green, and the one associated with a Neumann boundary condition in red. It is a straightforward approach to apply the Dirichlet boundary conditions, though it is by no means the only one.

<sup>†</sup>So a trivial optimization step is to compute only half of  $\mathbf{K}$ .

<sup>‡</sup>For interpolating bases such as the Lagrange polynomials, a Dirichlet boundary condition is often expressed directly in terms of nodal values.

Observe that, so far, we have made a few assumptions. First, the unknown solution  $u$  might not be an element of our subspace  $V_h$ , making  $u_h$  only an approximation. In fact, this is usually the case; it is known as the *discretisation error*. In addition, the same likely holds for the functions  $f$ ,  $b_D$  and  $b_N$ . In the next few sections, we will encounter additional error sources. The key to the proper application of FEA is to carefully balance these.

Let us take a closer look at the basis functions  $N_k(x, y)$ . Currently, they are *global* basis functions. However, as we have seen in Chapter 2, the basis functions used to define curves or surfaces usually have local support. In addition, in the setting of curves and surfaces, each basis function is associated with a control point. In the setting of FEM, we currently just have a region  $\Omega$ . We proceed to *partition* this region into *elements*  $K_l$ , which together form a *mesh*. The elements are usually triangular or quadrilateral, and can be linear or of higher degree. It is not uncommon that the mesh only approximates the domain (i.e.  $\cup_l K_l \neq \Omega$ ) – for example, consider a curved domain that is approximated by a mesh consisting of linear elements. This error source is known as the *geometry error*.

In the context of FEM, the control points of the mesh are referred to as *nodes*. The nodes are associated with *local* basis functions, which in a FEM setting are known as *shape functions*. It follows that the equivalent of surface patches in FEM are the elements. They can be parameterised on the unit triangle or unit square, which are referred to as the *reference element* (also known as the parent or master element).

Certainly, an important question is what type of element(s) to choose to partition  $\Omega$ . A common (and convenient) approach is to use *isoparametric* elements, which means that the basis selected for  $V_h$  is also used to define the elements  $K_l$ . In other words, the nodes then correspond to coefficient triples  $(x_k, y_k, u_k)$ . Of course, in the case of e.g. a rectangular domain, we could choose to use linear elements, while using a basis of higher degree to span a suitable  $V_h$ . The elements are then referred to as *subparametric*. The opposite case results in *superparametric* elements.

Assuming an isoparametric approach, a classical choice for triangular elements are the Lagrange elements (which include linear elements), see Section 2.2.6. Note that for these elements, the space spanned by the shape functions corresponds to the degree of the element. That is, for a quadratic triangular Lagrange element, the 6 shape functions span the space of quadratic polynomials that is also spanned by the power basis  $\{1, x, y, x^2, xy, y^2\}$ .

Lagrange elements are also often used for quadrilateral elements. However, observe that for e.g. a quadratic *quadrilateral* Lagrange element, the space of quadratic polynomials is only a subspace of the span of its basis functions. In terms of the power basis, we have  $\{x^2y, xy^2, x^2y^2\}$  as additional functions. Clearly, the resulting space is not complete in the sense that it covers all cubic or quartic polynomials, only some of them. There is an interesting alternative family of elements known as the *serendipity elements* that aims at removing most of the interior nodes<sup>†</sup> of an element while still spanning the complete space of degree  $d$  polynomials. For quadratic and cubic serendipity elements, all interior nodes can be removed, resulting in 8- and 12-node elements, respectively. For quartics and higher, however, there have to be one or more

<sup>†</sup> It is a common misconception that for *any* serendipity element all interior nodes can be removed.

interior points to satisfy symmetry conditions of an element (known as *spatial isotropy*) and to span the entire polynomial space. Serendipity elements are *compatible* with Lagrange elements, as both sets of shape functions reduce to univariate Lagrange functions on the boundary.

### 4.1.3 Quadrature and assembly

Once the region  $\Omega$  has been partitioned into elements  $K_l$ , we can move on to computing the stiffness matrix  $\mathbf{K}$ , mass matrix  $\mathbf{M}$  and boundary mass matrix  $\mathbf{B}$  (4.11). The integrals over  $\Omega$  and  $\Gamma_N$  are split into integrals over single elements, resulting in element matrices (i.e. local matrices)  $\mathbf{K}_l$ ,  $\mathbf{M}_l$  and  $\mathbf{B}_l$ . Now, instead of defining the shape functions for each element  $K_l$  individually and subsequently integrating in the physical domain  $(x, y)$ , we use the reference element  $\hat{K}$  and integrate in the parametric domain  $(\xi, \eta)$ . The map from the reference element to a physical one is defined as

$$x(\xi, \eta)|_{K_l} = \sum_k x_{l,k} R_k(\xi, \eta) = \mathbf{x}_l^T \mathbf{R}(\xi, \eta), \quad (4.13)$$

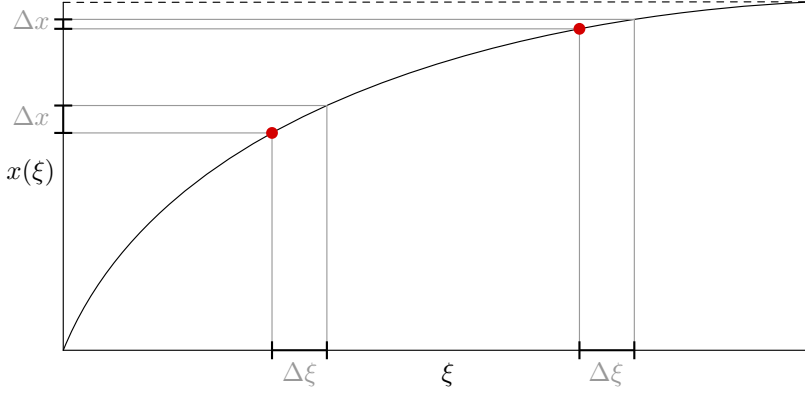
$$y(\xi, \eta)|_{K_l} = \sum_k y_{l,k} R_k(\xi, \eta) = \mathbf{y}_l^T \mathbf{R}(\xi, \eta), \quad (4.14)$$

with  $R_k(\xi, \eta)$  local shape functions in *parameter* space (i.e. shape functions defined on the reference element). Using shorthand notation we express this map as  $F_l$ , so that  $K_l = F_l(\hat{K})$ . Using the inverse of  $F_l$  and the isoparametric principle, we obtain

$$u_h(x, y)|_{K_l} = \sum_k u_{l,k} R_k(F_l^{-1}(x, y)) = \sum_k u_{l,k} L_k(x, y), \quad (4.15)$$

with  $L_k(x, y)$  local shape functions in *physical* space (i.e. shape functions defined on a single physical element).  $R_k(F_l^{-1}(x, y))$  is also referred to as the *push-forward* of a shape function on the parametric domain to the physical domain. The physical shape functions generally span a different space than the polynomial space spanned by the shape functions over the reference element due to *distortions* of the element (e.g. aspect ratio distortion, angular/parallelogram distortion and distortion due to curved boundaries). This clearly affects the approximation properties of the element —constant and linear functions can always be represented, but quadratic and higher-order degree functions might not. For instance, neither a quadratic nor a cubic serendipity element can reproduce all quadratic functions in the case of angular distortion; quadratic Lagrange elements still can, however in the case of distortion due to curved boundaries, also these elements lose the ability to fully represent quadratics [LB93; ZTZ05].

Now, recall that upon changing the domain of integration, the line-, area- or volume element of integration changes as well. This is best illustrated in the univariate case for the integration of a function  $g(x)$  with  $x = x(\xi)$ ; see Figure 4.1. The integral  $\int_D g(x) dx$  over some domain  $D$  is then equivalent to  $\int_{\hat{D}} g(x(\xi)) \frac{dx}{d\xi} d\xi$  over the appropriate domain  $\hat{D}$ .



**Figure 4.1:** Visualisation of the change of line element upon changing the domain of integration. Note that  $\Delta x \approx \frac{dx}{d\xi} \Delta\xi$ , i.e. it is a local linearisation of  $x(\xi)$  that approximates how much the increment (line piece)  $\Delta\xi$  should be scaled to be of the same length as the increment (line piece)  $\Delta x$ . It turns into  $dx = \frac{dx}{d\xi} d\xi$  when  $\Delta\xi$  becomes infinitesimally small.

In our bivariate setting, this generalises to

$$\Delta x \Delta y \approx \det \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix} \Delta \xi \Delta \eta = \det(\mathbf{J}) \Delta \xi \Delta \eta,$$

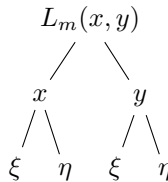
where  $\mathbf{J}$  is the Jacobian of  $F_l$ . It is a local linearisation of  $F_l$  that approximates how much the increment (parallelogram)  $\Delta \xi \Delta \eta$  should be scaled to cover the same area as the increment (square)  $\Delta x \Delta y$ . Note that, using (4.13) and (4.14), the Jacobian can be expressed as

$$\mathbf{J} = \begin{pmatrix} \mathbf{x}^T \\ \mathbf{y}^T \end{pmatrix} \begin{pmatrix} \frac{\partial \mathbf{R}(\xi, \eta)}{\partial \xi} & \frac{\partial \mathbf{R}(\xi, \eta)}{\partial \eta} \end{pmatrix}.$$

It follows that an entry of the element mass matrix  $\mathbf{M}_l$  can be calculated as

$$\int_{K_l} L_m(x, y) L_n(x, y) dx dy = \int_{\hat{K}} R_m(\xi, \eta) R_n(\xi, \eta) \det(\mathbf{J}) d\xi d\eta. \quad (4.16)$$

Next, we consider the element stiffness matrix  $\mathbf{K}_l$ , containing entries  $\nabla L_m (\nabla L_n)^T = \frac{\partial L_m}{\partial x} \frac{\partial L_n}{\partial x} + \frac{\partial L_m}{\partial y} \frac{\partial L_n}{\partial y}$ . In addition, consider the following tree diagram visualising the dependencies of  $L_m$ :



From the chain rule it follows that the gradient of  $L_m$  with respect to  $\xi$  and  $\eta$ , which we express as  $\nabla_{\xi\eta}$ , is

$$\nabla_{\xi\eta} L_m = \begin{pmatrix} \frac{\partial L_m}{\partial x} \frac{\partial x}{\partial \xi} + \frac{\partial L_m}{\partial y} \frac{\partial y}{\partial \xi} \\ \frac{\partial L_m}{\partial x} \frac{\partial x}{\partial \eta} + \frac{\partial L_m}{\partial y} \frac{\partial y}{\partial \eta} \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix} \begin{pmatrix} \frac{\partial L_m}{\partial x} \\ \frac{\partial L_m}{\partial y} \end{pmatrix} = \mathbf{J}^T \nabla_{xy} L_m, \quad (4.17)$$

so that  $\nabla_{xy} L_m = \mathbf{J}^{-T} \nabla_{\xi\eta} L_m$ . The result is that

$$\int_{K_l} \nabla_{xy} L_m (\nabla_{xy} L_n)^T dx dy = \int_{\hat{K}} \mathbf{J}^{-T} \nabla_{\xi\eta} R_m \left( \mathbf{J}^{-T} \nabla_{\xi\eta} R_n \right)^T \det(\mathbf{J}) d\xi d\eta. \quad (4.18)$$

An important aspect we have not yet touched upon is *how* to perform the actual integration. Although in the current setting — i.e. solving the BVP for Poisson's equation on a planar domain — (4.16) is polynomial and (4.18) is rational<sup>†</sup> and might therefore be integrated analytically, the general approach is to integrate numerically using a suitable set of quadrature rules. However, the latter might not yield exact integrals, and as such introduces a *quadrature error*.

A general approach to construct appropriate quadrature rules is to consider a space of functions that should be integrated exactly by them. For example, consider the space of univariate cubic polynomials  $\{1, x, x^2, x^3\}$  on the unit interval  $[0, 1]$ . The challenge is to find suitable points  $x_k$  with associated weights  $w_k$  such that for any function  $f(x)$  in this space, we have

$$\int_0^1 f(x) dx = \sum_k f(x_k) w_k. \quad (4.19)$$

An initial approach could consider two unknown points  $\{x_1, x_2\}$  and two unknown weights  $\{w_1, w_2\}$ , providing 4 DoFs to integrate any function in our 4-dimensional space. Along with the exact integrals for the basis functions, this results in the *nonlinear* system of equations

$$\begin{pmatrix} \int_0^1 1 dx \\ \int_0^1 x dx \\ \int_0^1 x^2 dx \\ \int_0^1 x^3 dx \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ x_1 & x_2 \\ x_1^2 & x_2^2 \\ x_1^3 & x_2^3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \end{pmatrix}. \quad (4.20)$$

Using an iterative approach with a reasonable initial guess (e.g. we require  $w_1 + w_2 = 1$ ) yields the symmetric solution  $(x_1, w_1) = (0.211324865405188, 0.5)$  and  $(x_2, w_2) = (0.788675134594813, 0.5)$ . Also note that this is a *minimal* (also called *Gaussian*) quadrature rule — it is not possible to integrate all functions in this space using only a single integration point  $x_1$  (e.g. consider a parabola with its root at  $x_1$ ).

<sup>†</sup>The inverse of  $\mathbf{J}$  is straightforward as it is a  $2 \times 2$  matrix, and results in the appearance of a factor  $1/\det(\mathbf{J})$ .

Unfortunately, for larger spaces (especially in the multivariate setting) this approach becomes increasingly more difficult. Luckily, there is an alternative known as the *Gauss–Legendre* rules. First, we construct a set of orthogonal polynomials on the unit interval, starting with 1 and subsequently considering powers of  $x$  that are made orthogonal to the other basis functions by means of the Gram–Schmidt procedure. This results in the Legendre polynomials  $P_k(x)$ <sup>†</sup>.

Next, we consider a degree  $2d - 1$  polynomial space whose functions  $f(x)$  we want to integrate exactly. Dividing  $f(x)$  by the Legendre polynomial function  $P_d(x)$  yields

$$f(x) = Q(x)P_d(x) + R(x), \quad (4.21)$$

where both  $Q(x)$  and  $R(x)$  are polynomials of (at most) degree  $d - 1$ . Writing  $Q(x)$  in terms of Legendre polynomials as  $Q(x) = \sum_k c_k P_k(x)$  and using the orthogonality property of the Legendre basis, we get

$$\int_0^1 f(x) dx = \int_0^1 \left( \sum_{k=0}^{d-1} c_k P_k(x) \right) P_d(x) + R(x) dx = \int_0^1 R(x) dx.$$

Finally, we express  $R(x)$  in terms of a degree  $d - 1$  Lagrange basis (see Section 2.2.6), though instead of using uniform nodes for the Lagrange polynomials, we use the  $d$  roots  $x_l$  of the degree  $d$  Legendre polynomial  $P_d(x)$  as nodes. We thus obtain

$$R(x) = \sum_{l=0}^{d-1} d_l \mathcal{L}_l^{d-1}(x),$$

where  $d_l \in \mathbb{R}$  are the coefficients assuming the role of control points (we are in the functional setting after all). Recall that a Lagrange curve interpolates its control points – in other words, we have  $d_l = R(x_l)$ . Although we do not know  $R(x)$ , from (4.21) it follows that  $R(x_l) = f(x_l)$  because  $P_d(x_l) = 0$ . We conclude that

$$\int_0^1 f(x) dx = \int_0^1 R(x) dx = \sum_{l=0}^{d-1} f(x_l) \int_0^1 \mathcal{L}_l^{d-1}(x) dx = \sum_{l=0}^{d-1} f(x_l) w_l, \quad (4.22)$$

showing that a degree  $2d - 1$  polynomial can be integrated using  $d$  integration points  $x_l$  (i.e. the roots of the Legendre polynomial  $P_d(x)$ ) and weights  $w_l$  (i.e. the integrals of the Lagrange polynomials  $\mathcal{L}_l^{d-1}(x)$ ).

In the bivariate setting, tensor-products of the Gauss–Legendre rules can be used to integrate polynomial functions defined on the unit square. The univariate weights are multiplied, which leads to the somewhat curious result that an  $a \times b$  rule can integrate all  $4ab$  basis functions spanning the space of  $(2a - 1) \times (2b - 1)$  degree polynomials using a mere  $3ab$  DoFs. For example, a  $2 \times 2$  rule suffices to integrate all 16 basis functions spanning the space of bicubic polynomials, which comes down to only 12 DoFs (i.e.  $2 \times 2$  different  $u$  and  $v$  coordinates of the integration points and an equal number of weights that are the *products* of the univariate weights).

<sup>†</sup>Or rather, it results in dilated and shifted Legendre polynomials, as they are commonly defined on the interval  $[-1, 1]$ .

For triangular domains the Gram-Schmidt procedure yields a set of bivariate orthogonal functions (i.e. on the unit triangle), but unfortunately these are not symmetric with respect to  $x$  and  $y$ . As such, they are not used — development of suitable quadrature relies on the generalisation of the approach taken in (4.20) and can be found in e.g. [Dun85].

With quadrature for polynomials now at our disposal, the integrals in (4.16) and (4.18) can be approximated. The typical approach is to use numerical integration that is exact for the polynomial shape functions, though due to the presence of the determinant of the Jacobian (and for the stiffness matrix, its inverse), the results are seldom exact. In the case of  $f$  and  $b_N$ , an alternative to their projection onto  $V_h$  and the use of the (boundary) mass matrix is to evaluate these functions directly at the quadrature points.

After computing all element matrices, they are *assembled* into the global matrices, sometimes expressed as

$$\mathbf{K} = \bigcup_l \mathbf{K}_l,$$

which from an implementation point of view comes down to proper bookkeeping of indices. The global numbering of nodes can be a significant aspect in solving the system, as it determines the bandwidth of the stiffness matrix (in other words, elements whose node indices are far apart influence the structure of the global matrix).

#### 4.1.4 Solving the linear system $\mathbf{K}u = q$

Finally, with the element matrices computed and assembled into global matrices, the linear system (4.10) can be solved. There are several observations to be made. First, note that  $\mathbf{K}$  is *sparse*, which follows from the fact that the shape functions have local support (i.e. only a few shape functions have support over an element  $E_l$ ). Second, recall that  $\mathbf{K}$  is symmetric. Together, this allows the use of special solvers.

Although for small systems a direct solver could be used, typically an iterative approach is applied. The solutions obtained might not be exact, but instead match a certain tolerance, resulting in the *solver error* (which might also occur when using direct solvers because of roundoff/precision errors). There is a large body of literature on the topic of solvers and related matters such as preconditioning, discussion of which is out of scope here; we refer to [Saa03; Vor03] for details.

#### 4.1.5 The patch test

The idea of the *patch test* is to check whether the approximation space spanned by the *physical* shape functions is *complete* in the sense that it can represent all constant and linear functions [ZTZ05]. It is established by running two sets of tests on a small mesh of elements referred to as the *patch* (not to be confused with a surface patch  $S(u, v)$ ) containing at least one interior node. In addition, the elements in the patch should not be scaled or dilated versions of the reference element. The first test considers rigid body motions by applying a translation (i.e.  $x$ ,  $y$  or  $x + y$ ) or rotation to all boundary nodes of the patch — if the interior node(s) match the expected value(s) for all possible

rigid body motions, this test is considered passed. The second test applies minimal Dirichlet boundary conditions to the patch so that it cannot translate or rotate upon the imposition of (Neumann) boundary conditions elsewhere. The latter are applied such that the analytic solution is linear, yielding a constant derivative. Historically, this was applied within the context of linear elasticity, imposing loads on certain boundary nodes resulting in a linear displacement and therefore a constant strain [IR72]. The test is considered passed if the resulting approximation  $u_h$  matches the expected linear function.

In theory, isoparametric elements automatically satisfy the patch test because the shape functions form a partition of unity. Clearly, this allows for the representation of constants. Moreover, since  $x = \sum_k x_k R_k$  and  $y = \sum_k y_k R_k$ , it follows that by using control points  $u_k = \alpha_0 + \alpha_1 x_k + \alpha_2 y_k$  we obtain

$$\begin{aligned} u_h &= \sum_k u_k R_k = \sum_k (\alpha_0 + \alpha_1 x_k + \alpha_2 y_k) R_k \\ &= \alpha_0 \sum_k R_k + \alpha_1 \sum_k x_k R_k + \alpha_2 \sum_k y_k R_k = \alpha_0 + \alpha_1 x + \alpha_2 y, \end{aligned}$$

which shows that indeed any constant or linear field  $u_h$  can be represented on the element (regardless of the type of distortion as mentioned above) [FB07]. As a side note, we mention that bi-quadratic precision requires at least biquartic elements [LB93].

In practice, the patch test for an isoparametric element depends on the choice of quadrature. If the selected rule is not accurate enough, the numerical integration might not yield a result that is within an acceptable tolerance of the solution, and as such actually fail the patch test. On the other hand, if the selected quadrature rule requires an enormous amount of integration points, it is probably not a practical rule; whether or not *efficiency* should be included as a criterion for passing the patch test is open for discussion.

We conclude with a note that there exist higher-order patch tests, which might be useful in the study of higher-order PDEs. The classic by Zienkiewicz et al. [ZTZ05] devotes an entire chapter to the matter.

#### 4.1.6 Error estimators and refinement

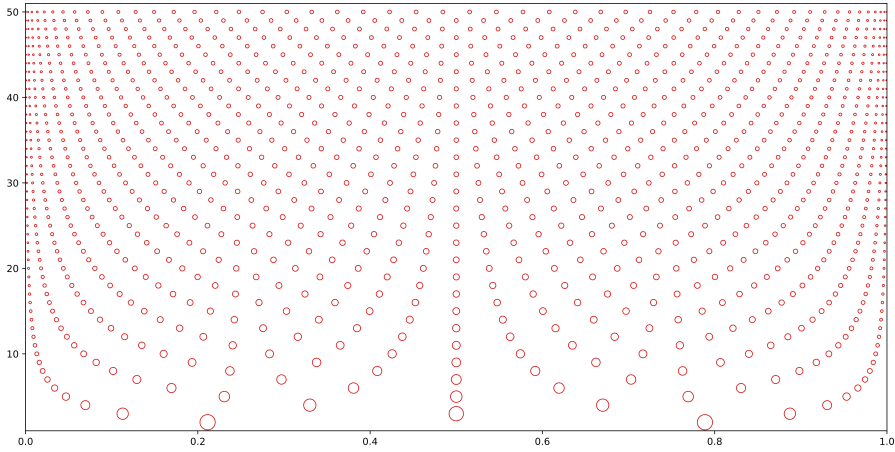
Once we have solved the linear system of equations and are provided with a solution (likely visualised in 3D using a suitable colour map as a post-processing step), how can we make sure that it is in fact a reasonable approximation? After all, there are quite a few error sources — for convenience, we have summarised them below:

- Modelling error (the PDE describing a physical phenomenon might be a simplification). In our case, we did not mention a specific application, and used Poisson's equation as starting point.
- Discretisation error (the space  $V_h$  usually does not contain the exact solution  $u$ ).
- Boundary condition and source term error (likewise,  $V_h$  might not contain  $b_D$ ,  $b_N$  or  $f$ ).



- Geometry error (the domain might be merely approximated by the union of elements, i.e. the mesh).
- Quadrature error (the numerical integration applied to compute the element matrices is usually not exact).
- Solver error (the iterative procedure for solving the linear system requires a termination condition, usually based on some tolerance).

In case of the first error source, we have no other option than to compare it to results of an actual experiment<sup>†</sup>. As such, we ignore this source in the remainder of the discussion. The following three error sources can intuitively be reduced by partitioning the domain into smaller elements, higher order elements or a combination of the two. This should likely be done only locally; we will get back to this below. Quadrature errors might be reduced by choosing a better quadrature rule, though this requires detailed knowledge of the integrands. Simply raising the degree of e.g. Gauss–Legendre quadrature does not always result in the desired improvement — the integrand is sampled at an increasing number of points which are *not* equally distributed on the domain and are *not* nested (see Figure 4.2), so local features of the integrand might not be detected. Gauss–Legendre effectively interprets the samples as coefficients (i.e. 1D control points in the functional setting) associated with Lagrange polynomials and integrates the resulting polynomial function exactly. Finally, solver errors might be ameliorated by setting a stricter tolerance or switching to a more suitable (e.g. more robust) solver.



**Figure 4.2:** Gauss–Legendre integration points for  $d \in [2, 50]$ . The corresponding weights  $w \in [0.0029, 0.5]$  are encoded in the diameters of the integration points.

In the case of elliptic PDEs (like our Poisson problem), so-called *a priori* error estimates can be obtained that are usually expressed as  $\|u - u_h\|_{H^1(\Omega)} \leq Ch^d |u|_{H^{d+1}(\Omega)}$ , with  $C$  a constant,  $h$  some measure on the mesh (e.g. the maximal element diameter)

<sup>†</sup> Clearly, an alternative and also follow-up is to derive the mathematical model of the phenomenon again based on theoretical or empirical insights.

and  $d$  the degree of the elements. Note that  $\|\cdot\|_{H^1(\Omega)}$  is the norm associated with the Hilbert space  $H^1(\Omega)$  and  $|\cdot|_{H^{d+1}(\Omega)}$  the semi-norm associated with the Hilbert space  $H^{d+1}(\Omega)$  [Joh12]; similar expressions based on norms in different spaces (e.g.  $L^2$ ) are available [BS08]. This estimate tells us something about the *rate of convergence* when the elements are uniformly refined or increased in degree. This might influence our initial choice of element type, as well as our refinement strategy.

Unfortunately, an a priori error estimate does not provide us with *local* information about the result that we obtain upon solving the linear system. However, we *do* know that it is the best approximation possible. This follows by taking the difference of the weak form before and after discretising it:

$$\int_{\Omega} \nabla(u - u_h) \cdot \nabla v_h \, d\Omega = 0 \quad \forall v_h \in V_h,$$

with  $v = v_h$  also before discretising, which is a valid choice as  $V_h \subset H^1(\Omega)$ . It is known as *Galerkin orthogonality* and shows that the error  $u - u_h$  is orthogonal to the space  $V_h$  (with respect to the bilinear form on the left-hand side).

Finally, this brings us to *a posteriori* error estimators. The general idea is to construct an error estimator (with both a lower- and an upper bound) that can be split into contributions over single elements. Considering the original PDE, a common approach is to use the residual  $f(x, y) + \Delta u_h(x, y)$  as the main ingredient to compute element refinement indicators. Subsequently, different *marking strategies* can be used to decide which elements to refine (e.g. a certain percentage of the elements or all elements whose indicators are within a certain percentage of the average or maximal element indicator).

Clearly, the above is only a very concise outline of error estimation — for more details, we refer to e.g. [AO11].

Once it has been decided which regions of the mesh need to be refined, we can proceed to locally enrich our space  $V_h$ . Regardless of the type of refinement (i.e. *h*-refinement which means splitting elements or *p*-refinement which means increasing the degree of elements), basis functions are added to  $V_h$  (usually replacing others). It is important that this is done in a *hierarchical* or *nested* fashion, that is, that the previously obtained approximation is still an element of  $V_h$  following its enrichment; this way, the new approximation cannot be worse than the one preceding it.

In the case of a mesh composed of linear triangular elements, the typical workflow is to split elements by introducing new nodes and connecting them to surrounding nodes in a way that no *hanging nodes* are introduced (i.e. corner nodes of elements that live on edges of other elements). Care should be taken not to introduce elements with small angles. In addition, in regions with a very low error, one could experiment with merging elements (i.e. coarsening instead of refining), though this usually violates the hierarchical condition mentioned above. Note that for linear elements, the Laplacian of our approximation,  $\Delta u_h(x, y)$ , vanishes, which requires the use of a different indicator.

For higher-order elements, we can follow a similar approach, though remember that new nodes have to be introduced in such a way that the former solution can still be constructed. This essentially comes down to knot-insertion like we discussed in Section 2.3.1. For these higher-order elements, we can use a residual-based refinement

indicator. However, classical elements are typically only  $C^0$  across element boundaries, which means that the Laplacian of the shape functions results in Dirac delta functions at the boundaries. This results in the appearance of an additional term in the computation of the refinement indicator commonly referred to as the *jump term*. As a side note, this illustrates how convenient the weak form is, as it only considers integrals of first-order partial derivatives of  $u_h(x, y)$ . Using  $C^0$  shape functions, these partial derivatives are square-integrable, which is in fact the reason for selecting the space  $H^1(\Omega)$  when we formulated the BVP. When considering a fourth-order PDE containing e.g. the biharmonic operator, the weak form also considers second-order partials, which requires  $C^1$  shape functions and motivates the use of the space  $H^2(\Omega)$ . We will get back to this in the next section.

Alternatively, the order of the elements can be increased. The standard example is to consider linear triangular elements and degree-elevate them to become quadratic. As this is typically done locally, so-called *transition elements* are required to connect the linear parts of the mesh with the quadratic parts.

The next section will consider  $h$ -refinement in more detail for the specific case of using Catmull–Clark subdivision splines as basis functions.

## 4.2 Isogeometric analysis (IgA)

With the basics of FEM introduced, the notion of *isogeometric analysis* (IgA) is now straightforward to define — it simply reverses the concept of isoparametric elements in the sense that instead of constructing the mesh based on the choice of the space  $V_h$  to approximate  $u(x, y)$ , the approximation space is now spanned by the basis functions describing the geometry of the object that we want to run FEA on. Because of this, the *geometry error* usually vanishes. At the same time, meshing is now straightforward as the elements directly correspond to surface patches or subdivisions thereof. This can save quite a bit of time compared to classical FEA, where meshing can be a time-consuming aspect of the entire process [CHB09], especially in the trivariate case. Moreover, in a classical setting, re-meshing and refinement often requires manual intervention, even more so when no parameterisation of the geometry is available (the object of interest might be defined using e.g. constructive solid geometry). Evidently, an iterative workflow where a geometry (e.g. a mechanical part) is repeatedly modified based on numerical simulation results provided by FEA benefits from a closer link between CAGD and FEA. Shape- and topology optimization are exemplary applications that fit into this framework.

From the point of view of approximation, IgA enjoys the favourable properties of the splines that are used to model the geometries of interest. These are often  $C^1$  or higher-order continuous and carry over their smoothness to the approximation  $u_h(x, y)$ . This high-order continuity across element boundaries allows for more faithful representations of physical quantities. Taking linear elasticity as an example, the strain  $\epsilon$  and stress  $\sigma$  depend on the first-order derivatives of the displacement, which is usually the unknown field solved for. With classical  $C^0$  Lagrange elements, the resulting strain and stress fields are therefore  $C^{-1}$  — using IgA instead allows for continuous and possibly smooth representations of these fields. Additionally, the high-order continuity of the basis functions facilitates the consideration of higher-order PDEs including e.g.

fourth-order partial derivatives. The larger overlap of spline basis functions compared to classical  $C^0$  ones, along with their non-negativity and (therefore) improved stability, typically results in smaller stiffness matrices (i.e. there is a higher efficiency per node) which might also have a lower condition number compared to the classical approach.

Finally, refinement relations (such as knot-insertion or the two-scale relation) and degree-elevation known from CAGD can be directly applied to implement  $h$ - and  $p$ -refinement<sup>†</sup>.

It should also be mentioned that IgA comes with certain drawbacks. A practical example is the imposition of boundary conditions, particularly the Dirichlet ones. Recall that, unlike Lagrange basis functions, splines generally do not interpolate their control points. As such, nodal values do not reflect physical values of the field  $u_h(x, y)$  at the nodes. We already mentioned that boundary conditions can be projected onto  $V_h$ , but did not yet elaborate. In short, there are several approaches to (weakly) enforce essential boundary conditions, including various variants of least-squares fitting [HCB05; MGT11], the use of Lagrange multipliers or penalty methods [Bab73a; Bab73b] and Nitsche's method [EDH10].

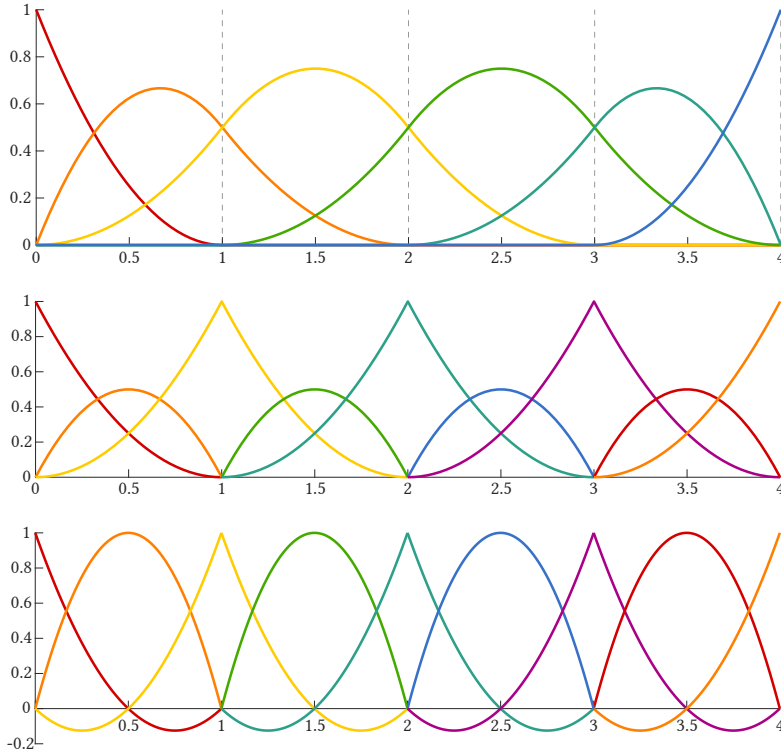
An apparent disadvantage of IgA might be the multitude of spline types that can be used to model an object, which would result in an equal number of different element types. However, recall that a polynomial spline curve can always be expressed as a sequence of connected Bézier curves (and therefore the splines themselves as combinations of Bernsteins). In the context of IgA this is referred to as *Bézier extraction* or *Bézier decomposition* [Bor+11; Sco+11]. It is a useful shortcut to implement IgA, and usually the only way to incorporate IgA in existing software packages as a user. It is applied only during the computation of the element matrices  $K_l$ . The idea is to consider *virtual* Bézier elements — such that all elements have a common basis — and use element extraction matrices to represent the relevant parts of the B-spline basis on each element. Note that the approach does *not* actually use the Bézier representation of the object to perform the analysis, as doing so would result in a  $u_h(x, y)$  that is merely  $C^0$  and thereby negate one of IgA's main benefits. Of course, instead of Bernstein polynomials, any other polynomial basis can be used for the same purpose. With Lagrange elements being a popular choice in classical FEA, *Lagrange extraction* [SRN16] might seem a more natural choice<sup>‡</sup>.

To summarise Bézier extraction and Lagrange extraction, Figure 4.3 shows three bases for a univariate example with 4 elements. Both the Bézier and Lagrange representations are readily obtained from the B-spline basis through repeated knot-insertion — in the former case it results in the Bernstein basis, whereas in the latter case it provides the nodal values that can be multiplied by the Lagrange basis functions.

Finally, we use the example from Figure 4.3 to compare the stiffness matrices resulting from IgA and classical FEA, respectively. The dimension of the *element* stiffness matrices is not affected by changing the approach and is  $3 \times 3$  in both cases, though the amount of overlap differs per approach. For IgA, we obtain a  $6 \times 6$  stiffness matrix,

<sup>†</sup>In addition to  $h$ - and  $p$ -refinement, in IgA there is another type of refinement referred to as  $k$ -refinement; it is a combination of order-elevation followed by knot-insertion (note that these are not commutative).

<sup>‡</sup>The notion of Bézier decomposition had just been published back when I started with my MSc internship, which involved the implementation of IgA in a commercial FEA package [Bar12a]. Being used to classical FEA as a student, I considered using Lagrange extraction instead, but ultimately did not pursue it.



**Figure 4.3:** Univariate non-uniform quadratic  $C^1$  B-spline basis associated with  $\Xi = [0, 0, 0, 1, 2, 3, 4, 4, 4]$  resulting in 4 elements connecting with  $C^1$  continuity (top). The quadratic  $C^0$  Bernstein and Lagrange bases are visualised as well (middle and bottom, respectively). Note that the B-spline basis consists of only 6 basis functions, whereas the Bernstein and Lagrange bases both contain 9 basis functions.

whereas FEA yields a  $9 \times 9$  stiffness matrix; see (4.23). The bandwidth of  $d + 1 = 3$  directly follows from the size of the element stiffness matrices (note that in the bi- or trivariate case we do not have such a simple relation because of the ordering of nodes). The  $6 \times 6$  matrix was also used in (4.12) with a Dirichlet boundary condition imposed on the first node and a Neumann boundary condition on the last one.

$$\begin{aligned}
 K = & \begin{pmatrix} \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \boxed{\begin{matrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{matrix}} \end{pmatrix} & K = \begin{pmatrix} \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \begin{pmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{pmatrix} \end{pmatrix} \quad (4.23)
 \end{aligned}$$

Following this brief introduction to IgA with its advantages and drawbacks, a historical note is in order. The insight that splines and FEA go well together predates the notion of IgA; a good example is the use of B-splines in FEA [Höl03], and in our context, the use of subdivision splines in FEA [COS00; CO01]. An overview of precursors of IgA is given in [Pro19]. Nevertheless, it is undeniable that the publication of [HCB05; CHB09] is largely responsible for the considerable interest in spline-based numerical methods for PDEs nowadays.

#### 4.2.1 Subdivision-based IgA

Let us now move our focus to subdivision splines in the context of IgA. Back when I started working on this topic, the existing literature was modest. In addition to the work of Fehmi Çırak and co-authors [COS00; CO01; Cir+02] using Loop subdivision, there were two sets of publications based on Catmull–Clark — Anna Wawrzinek’s work [WHP11] and Daniel Burkhart’s work [BHU10; Bur11] — and one publication using Doo–Sabin [DSO12]. Nowadays, the literature regarding subdivision-based IgA is much more extensive — some of it is referred to in the relevant sections below.

We chose to work with the Catmull–Clark scheme, mainly because of its prevalent use in the entertainment industry. Following an MSc exploratory project on planar subdivision-based IgA [Bar12b] late 2012 (written in article style but never submitted as such), I moved on to my MSc thesis on the same subject [Bar13] (initially with the ambitious thought to include subdivision solids, which quickly proved infeasible). Following the implementation of several test cases and benchmarks, the necessity for better quadrature rules became clear. In addition, it was evident that the approach would benefit from a local refinement approach.

Evidently, the main ingredient of subdivision-based IgA are the subdivision splines  $\varphi_{n,m}(u, v)$  and eigenfunctions  $\psi_{n,m}(u, v)$ , see Section 3.2.2. Recall that for Catmull–Clark subdivision, these are defined as in (3.47) and (3.48), which we repeat here for convenience:

$$\varphi_{n,m}(u, v) \Big|_{\Omega^{k,l}} = \mathbf{M}_{44}^T(t_{k,l}(u, v)) X_n^k \bar{\mathcal{A}}_n^S V_n \Lambda_n^{l-1} z_{n,m}, \quad (4.24)$$

$$\psi_{n,m}(u, v) \Big|_{\Omega^{k,l}} = \lambda_{n,m}^{l-1} \mathbf{M}_{44}^T(t_{k,l}(u, v)) X_n^k \bar{\mathcal{A}}_n^S \vec{w}_{n,m}. \quad (4.25)$$

The bilinear maps  $t_{k,l}(u, v)$  are defined as

$$t_{1,l} = (2^l u - 1, 2^l v), \quad t_{2,l} = (2^l u - 1, 2^l v - 1), \quad t_{3,l} = (2^l u, 2^l v - 1). \quad (4.26)$$

Furthermore, the level  $l$  is determined as  $l = \text{floor}(-\log_2(\max(u, v))) + 1$ . It follows that the eigenfunctions adhere to the scaling relation

$$\psi_{n,m}(u/2, v/2) = \lambda_{n,m} \psi_{n,m}(u, v). \quad (4.27)$$

Given the control net for a Catmull–Clark subdivision surface with at most one EV per face, each quad corresponds to a surface patch  $S(u, v)$ , which also acts as an element. It is defined by  $2n + 8$  control points and their associated subdivision splines, where  $n$  refers to the valency of the EV — in the case of a bicubic B-spline patch,  $n$  defaults to 4. By construction,  $x$ ,  $y$  and  $z$  are already expressed in terms of subdivision

splines. Reversing the isoparametric principle,  $u_h$  is then also expressed in terms of  $\varphi_{n,m}(u, v)$ . Eventually, the computation of the element mass- and stiffness matrices as formulated in (4.16) and (4.18) requires the integrals of products of subdivision splines and integrals of products of first-order partial derivatives of subdivision splines, in the latter case pre-multiplied by the inverse of the transpose of the Jacobian.

Now, recall that the subdivision splines are composed of an infinite number of piecewise bicubic layers, or in other words, are infinitely piecewise bicubic. Integrating them might therefore appear challenging. However, observe that — restricted to a tile  $\Omega^{k,l}$  — any  $\varphi_{n,m}(u, v)$  can be integrated exactly by integrating the bicubic B-splines  $\mathbf{M}_{44}$ , which is readily accomplished using a  $2 \times 2$  Gauss–Legendre quadrature rule with weights  $w_G$  and points  $(s_G, t_G)$ . The bilinear map  $t_{k,l}$  (mapping the tile to the unit square) results in a constant Jacobian of  $\left(\frac{1}{4}\right)^l = \frac{1}{4} \cdot \left(\frac{1}{4}\right)^{l-1}$ . As such,

$$\int_{\Omega^{k,l}} \varphi_{n,m}(u, v) d\Omega = \frac{1}{4} z_{n,m}^T \left(\frac{1}{4} \Lambda_n\right)^{l-1} (X_n^k \bar{\mathcal{A}}_n^S V_n)^T \sum_{G=1}^4 w_G \mathbf{M}_{44}(s_G, t_G).$$

Next, we can integrate over all three tiles in a single level  $l$  simply by adding the extraction matrices; we define  $\mathcal{X}_n = X_n^1 + X_n^2 + X_n^3$ . Finally, to integrate over all levels  $l \in [1, \infty]$  at once, observe that

$$\lim_{r \rightarrow \infty} \left(\frac{1}{4} \Lambda_n\right)^0 + \left(\frac{1}{4} \Lambda_n\right)^1 + \dots + \left(\frac{1}{4} \Lambda_n\right)^r = \left(I - \frac{1}{4} \Lambda_n\right)^{-1}. \quad (4.28)$$

In other words, the sum of powers of  $\left(\frac{1}{4} \Lambda_n\right)$  forms a geometric series, which converges because the largest entry in  $\Lambda_n$  is the dominant eigenvalue 1, which is scaled by  $\frac{1}{4}$ . We therefore obtain

$$\int_{\Omega} \varphi_{n,m}(u, v) d\Omega = \frac{1}{4} z_{n,m}^T \left(I - \frac{1}{4} \Lambda_n\right)^{-1} (\mathcal{X}_n \bar{\mathcal{A}}_n^S V_n)^T \sum_{G=1}^4 w_G \mathbf{M}_{44}(s_G, t_G). \quad (4.29)$$

Only after deriving the above, I found out that it had already been discovered before and was hiding in an appendix of [HKD93]; apparently, I was not the only one who initially overlooked it<sup>†</sup>.

When considering integrals of  $p^{th}$ -order partial derivatives of the subdivision splines w.r.t.  $u$  or  $v$ , the only differences in (4.29) are the appearance of an additional factor  $2^{pl} = 2^p \cdot 2^{p(l-1)}$  due to the map  $t_{k,l}$ , and the replacement of  $\mathbf{M}_{44}$  by the appropriate partial derivative of the bicubic B-splines. For  $p = 1$  it is straightforward to see that the geometric series converges. For  $p = 2$  it also converges — the dominant eigenvalue 1 is associated with the constant eigenfunction  $\psi_{n,0}(u, v)$ , whose partial derivatives evidently vanish. The subdominant eigenvalue  $\lambda_n$  is associated with the eigenfunctions  $\psi_{n,1}(u, v)$  and  $\psi_{n,n-1}(u, v)$ , whose second-order partials do *not* vanish because they are not linear using Stam’s parameterisation (which follows directly from the scaling relation (4.27)). Given that  $\lambda_n > \frac{1}{2}$  for  $n > 4$ , the series therefore does *not* converge for  $p \geq 3$  for these valencies.

<sup>†</sup>Personal correspondence with Jörg Peters.

In the case the product of two subdivision splines, we now need to integrate the  $16 \times 16$  matrix of products of bicubic B-splines. This results in bisextic functions, which we integrate exactly using a  $4 \times 4$  Gauss–Legendre rule. For shorthand notation, we introduce

$$\mathcal{C}_n^k = (X_n^k \bar{\mathcal{A}}_n^S V_n)^T \left( \sum_{G=1}^{16} w_G \mathbf{M}_{44}(s_G, t_G) \mathbf{M}_{44}^T(s_G, t_G) \right) (X_n^k \bar{\mathcal{A}}_n^S V_n), \quad (4.30)$$

so that we can express the integral of the product of subdivision splines  $\varphi_{n,a}$  and  $\varphi_{n,b}$  over a tile  $\Omega^{k,l}$  as

$$\int_{\Omega^{k,l}} \varphi_{n,a}(u, v) \varphi_{n,b}(u, v) d\Omega = \frac{1}{4} z_{n,a}^T \left( \frac{1}{2} \Lambda_n \right)^{l-1} \mathcal{C}_n^k \left( \frac{1}{2} \Lambda_n \right)^{l-1} z_{n,b}, \quad (4.31)$$

where the constant Jacobian has been split as  $\left(\frac{1}{4}\right)^l = \frac{1}{4} \cdot \left(\frac{1}{2}\right)^{l-1} \cdot \left(\frac{1}{2}\right)^{l-1}$  so that it can be distributed, resulting in  $\frac{1}{2} \Lambda_n$  on both sides. Note that we can no longer simply replace  $X_n^k$  by  $\mathcal{X}_n$  as this would result in cross-terms. Instead, we choose to integrate over the tiles  $\Omega^{k,l}$  for all levels  $l$  for each  $k$  individually. Introducing another shorthand notation  $\Gamma_n = \frac{1}{2} \Lambda_n$ , we obtain

$$\mathcal{S}_n^k = \lim_{r \rightarrow \infty} \Gamma_n^0 \mathcal{C}_n^k \Gamma_n^0 + \Gamma_n^1 \mathcal{C}_n^k \Gamma_n^1 + \cdots + \Gamma_n^r \mathcal{C}_n^k \Gamma_n^r = \mathcal{C}_n^k + \Gamma_n \mathcal{S}_n^k \Gamma_n. \quad (4.32)$$

Writing the above expression per entry  $(i, j)$  of  $\mathcal{S}_n^k$ , we get

$$[\mathcal{S}_n^k]_{ij} = [\mathcal{C}_n^k]_{ij} + [\Gamma_n]_{ii} [\mathcal{S}_n^k]_{ij} [\Gamma_n]_{jj} = \frac{[\mathcal{C}_n^k]_{ij}}{1 - [\Gamma_n]_{ii} [\Gamma_n]_{jj}}. \quad (4.33)$$

We conclude that

$$\int_{\Omega} \varphi_{n,a}(u, v) \varphi_{n,b}(u, v) d\Omega = \frac{1}{4} z_{n,a}^T \left( \sum_{k=1}^3 \mathcal{S}_n^k \right) z_{n,b}. \quad (4.34)$$

Note that (4.29) can also be expressed in this way, though the original expression is more compact. In fact, triple- and higher-order products can be expressed in a similar manner, although this requires the use of tensors (i.e. multidimensional matrices). We discuss this in Appendix C.

For the product of two  $p^{th}$ -order partial derivatives of the subdivision splines, we now obtain a factor  $2^p \cdot 2^{p(l-1)}$  on both sides, and the  $16 \times 16$  matrix of bisextic functions is replaced by a matrix containing the appropriate derivatives. For  $p = 1$ , the result is integrable (using the argument considering the vanishing partials of the constant eigenfunction), which shows that using Stam's parameterisation, the subdivision splines are elements of the Hilbert space  $H^1$ . However, for  $p = 2$ , the result is *not* integrable (following the argument that the second-order partials of  $\psi_{n,1}$  and  $\psi_{n,n-1}$  do not vanish), which means that the subdivision splines are *not* in  $H^2$  using this parameterisation.

Let us take a small detour to consider alternative partitions and parameterisations. Instead of Stam's partition of the unit square  $\Omega$  (using dyadic splits of the domain), we



could partition it based on the value of the subdominant eigenvalue  $\lambda_n$ , see Figure 4.4. Using modified bilinear maps  $t_{k,l}(u, v)$ , we then obtain the scaling relation

$$\psi_{n,m}(\lambda_n u, \lambda_n v) = \lambda_{n,m} \psi_{n,m}(u, v).$$

However, also in this case the resulting  $\psi_{n,1}$  and  $\psi_{n,n-1}$  are not linear, which is readily established by visual inspection of their first-order partials. Note that the new scaling relation is a necessary condition to obtain linear functions, but not a sufficient one.

An alternative is to parameterise the eigenfunctions on a sector of the image of the characteristic map [WW01]. Recall that the characteristic map is defined as the limit surface using the right eigenvectors  $\vec{w}_{n,1}$  and  $\vec{w}_{n,n-1}$  as the  $\xi$ - and  $\eta$ -ordinates of the control points. That is,  $(\xi, \eta) = \chi_n(u, v) = (\psi_{n,1}(u, v), \psi_{n,n-1}(u, v))$ . Extending the control points to  $(\xi, \eta, \zeta) \in \mathbb{R}^3$ , using one of these eigenvectors as  $\zeta$ -ordinate, we obtain meshes  $(\vec{w}_{n,1}, \vec{w}_{n,n-1}, \vec{w}_{n,1})$  and  $(\vec{w}_{n,1}, \vec{w}_{n,n-1}, \vec{w}_{n,n-1})$  defining the linear eigenfunctions  $\zeta = \xi$  and  $\zeta = \eta$ . Note that the required scaling relation holds by construction over the entire domain (each subdivision of the mesh – composed of eigenvectors – effectively results in a new spline ring, which is a copy of the previous one scaled by  $\lambda_n$ ). Re-formulating the scaling relation as

$$\psi_{n,m}(\xi, \eta) = \lambda_{n,m} \psi_{n,m}(\xi/\lambda_n, \eta/\lambda_n),$$

note that its first-order partial derivative with respect to  $\xi$

$$\frac{\partial \psi_{n,m}(\xi, \eta)}{\partial \xi} = \frac{\lambda_{n,m}}{\lambda_n} \frac{\partial \psi_{n,m}(\xi/\lambda_n, \eta/\lambda_n)}{\partial \xi}$$

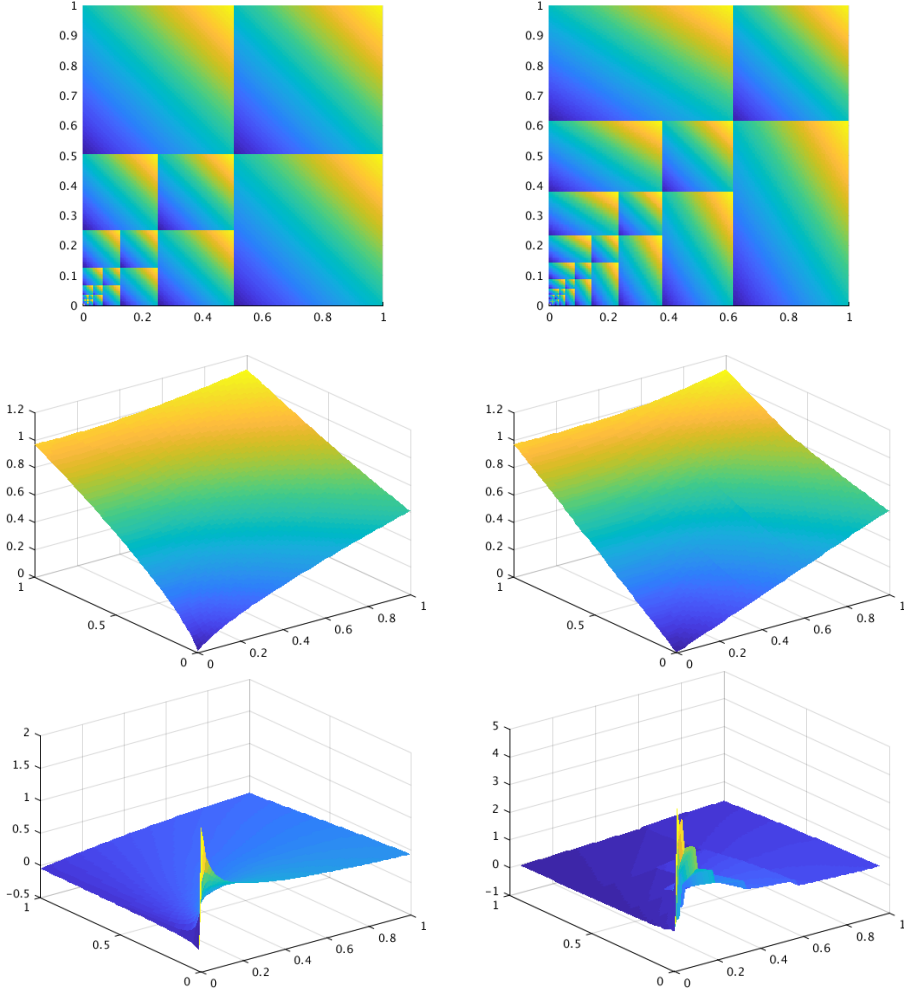
suggests constant first-order partials for the eigenfunctions associated with the  $\lambda_n$  (as expected – we just showed that these are linear on  $\chi_n$ ); mutatis mutandis for  $\eta$ . Their second- and higher-order partials vanish, suggesting that the subdivision splines are in  $H^2$ . In [RS01] it is shown that this is indeed the case.

Furthermore, we can now define our surface patches as  $S_\chi(\xi, \eta) = S(\chi_n^{-1}(\xi, \eta))$ , see Figure 4.5.

Also this approach comes with a drawback, as there is no closed-form expression for the inverse of the characteristic map  $\chi_n^{-1}(\xi, \eta)$ . However, using an iterative approach, it can be inverted point-wise, as explained in [BZ04; WP16].

The above observations allow us to sketch some informal proofs. First, the multiplicities of the eigenvalues  $\lambda_{n,m}$  of a subdivision scheme naturally imply the desired order of their associated eigenfunctions  $\psi_{n,m}$ . That is, the dominant eigenvalue should be associated with a constant eigenfunction, the subdominant eigenvalues  $\lambda_n$  with linearly independent linear ones, and the subsubdominant eigenvalues  $\mu_n$  with linearly independent quadratic ones. In addition, the corresponding right eigenvectors  $\vec{w}_{n,m}$  should be effective. In this view, the *effective* multiplicities of  $\lambda_n$  and  $\mu_n$  should clearly not exceed two and three, respectively, and in fact match these quantities exactly.

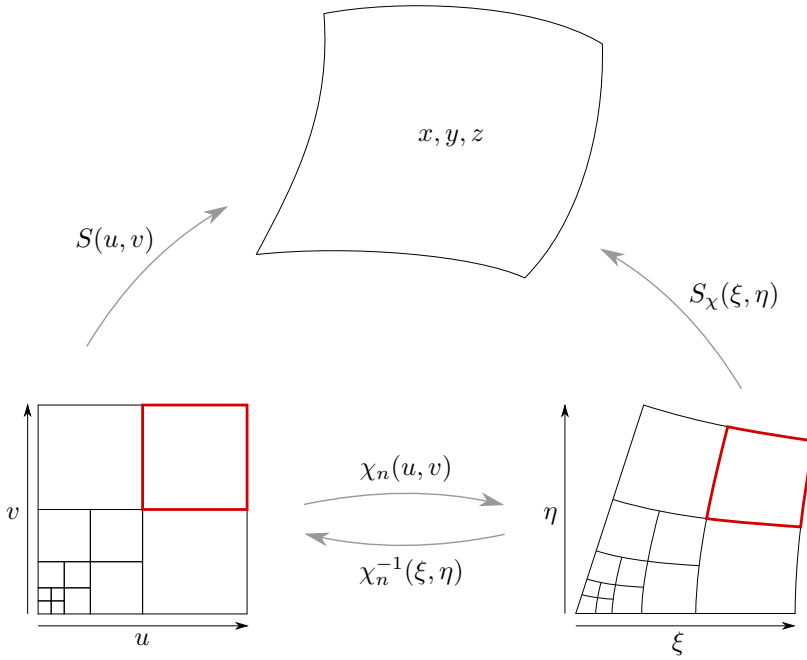
Next, the two eigenfunctions associated with the two subdominant eigenvalues should form a mesh (the natural configuration) such that its associated limit surface (the characteristic map  $\chi_n$ ) is injective and regular. This means that it can be used as a domain to parameterise the eigenfunctions themselves. Though  $\psi_{n,0}$  is a constant function regardless of the parameterisation domain, whether  $\psi_{n,1}$  and  $\psi_{n,n-1}$  are linear or



**Figure 4.4:** Two different partitions of the domain  $\Omega$  (top). On the left we have Stam's dyadic partition, on the right we have the lambda-partition using  $\lambda_8$ . The images of the bilinear maps  $t_{k,l}(u, v)$  are shown. For each partition, we visualise the eigenfunction  $\psi_{8,1}(u, v)$  (middle) and its first-order partial derivative with respect to  $u$  (bottom). Note that the latter cannot be evaluated at the origin as it is unbounded there. Stam's parameterisation yields subpatches with  $C^2$  continuity, but the resulting function is not linear. The alternative parameterisation results in  $C^{-1}$  continuity between the subpatches when using (modified) bilinear maps  $t_{k,l}(u, v)$ , though adheres to the necessary scaling relation.

not clearly depends on the choice of parameterisation. On  $\chi_n$  they are, which means that their second- and higher-order partials vanish. As a result, the subdivision splines  $\varphi_{n,m}$  are in  $H^2$ , which — for (infinitely) piecewise polynomial functions — implies that they are  $C^1$  continuous.

Now, with regard to the three eigenfunctions associated with  $\mu_n$ , consider the scal-



**Figure 4.5:** The definition of a surface patch  $S(u, v) \in \mathbb{R}^3$  using Stam's partition of the unit square  $\Omega$ , and its reparameterisation  $S_\chi(\xi, \eta)$  based on the (inverse of the) characteristic map  $\chi_n$ .

ing relation

$$\psi_{n,m}(\xi, \eta) = \mu_n \psi_{n,m}(\xi/\lambda_n, \eta/\lambda_n).$$

Taking its  $p^{\text{th}}$ -order partial derivative with respect to  $\xi$  or  $\eta$  results in an additional factor  $1/\lambda_n^p$ . For  $p = 2$  the resulting factor  $\mu_n/\lambda_n^2$  determines the curvature behaviour around EVs of the resulting limit surface. It also suggests that for  $\mu_n = \lambda_n^2$ , the second-order partials of these three eigenfunctions are constant. If we can get them to be quadratic on  $\chi_n$ , their third- and higher-order partials vanish. Assuming that all other eigenvalues have (absolute) values less than  $\mu_n$  should then result in subdivision splines that are in  $H^3$ , which can then be used to model  $C^2$  continuous limit surfaces.

Finally, observe that setting  $\lambda_n = \frac{1}{2}$  for all valencies  $n \geq 3$  not only results in a natural contraction rate of the spline rings around an EV for a binary scheme (as mentioned in the previous chapter), but also makes the scaling relation for Stam's parameterisation (4.27) the one required for linear  $\psi_{n,1}$  and  $\psi_{n,n-1}$ . It results in bounded first-order partials of the eigenfunctions; setting  $\mu_n = \lambda_n^2 = \frac{1}{4}$  ensures bounded first- and second-order partials for the three eigenfunctions associated with  $\mu_n$ . Recalling the  $C^2$  continuity between subpatches in Stam's parameterisation, this suggests that – with these eigenvalues – the subdivision splines would be in  $H^2$ . Note that the Doo-Sabin scheme and our new scheme based on half-box splines satisfy this distribution of eigenvalues by construction, which means that although they are both  $C^1$  continuous, they enjoy these satisfactory properties out-of-the-box.

Ultimately, the occurrence of (inverses of) Jacobians in FEA integrands prevents us from applying these exact integration approaches in the context of IgA (still, the exact approaches have their use, as demonstrated later on). We are left with various alternatives, including the following:

- 1.) Using tensor-product Gauss–Legendre on the unit square  $\Omega$ . Simply increasing the order of the quadrature rule samples the integrand at an increasing number of positions (see Figure 4.2). However, there is no indication that this is actually useful in the case of subdivision splines [Bar13]. Sampling more points away from the EV should not be necessary as the polynomial pieces are relatively large in these regions. Near the EV matters are reversed as the pieces eventually get infinitesimally small here, requiring more points.
- 2.) Using tensor-product Gauss–Legendre on individual tiles  $\Omega^{k,l}$  [Sco11; NKP14]. This results in an increasing density of integration points towards the EV, which remedies the shortcoming of the above approach. However, there are an infinite number of levels – in practice the integration can of course only be applied to a finite number of them. At which level to truncate seems to be an empirical matter.
- 3.) Deriving quadrature rules for *macro patches* (i.e. patches spanning several tiles) that are more efficient than the per-tile approach. In Section 4.2.2 we show that this can be done because of the continuity between subpatches. Like the per-tile approach, we need to truncate the integration after a certain number of levels.
- 4.) Deriving new quadrature rules for subdivision splines by generalising the approach demonstrated in (4.20). In Section 4.2.4 we look into this.

Two alternative approaches referred to as *barycenter quadrature* and *mid-edge quadrature* are compared to 1.) and 2.) in [Jüt+16] in the context of Loop subdivision. Their conclusion is that 2.) is the most robust option, albeit rather expensive because of the high number of integration points.

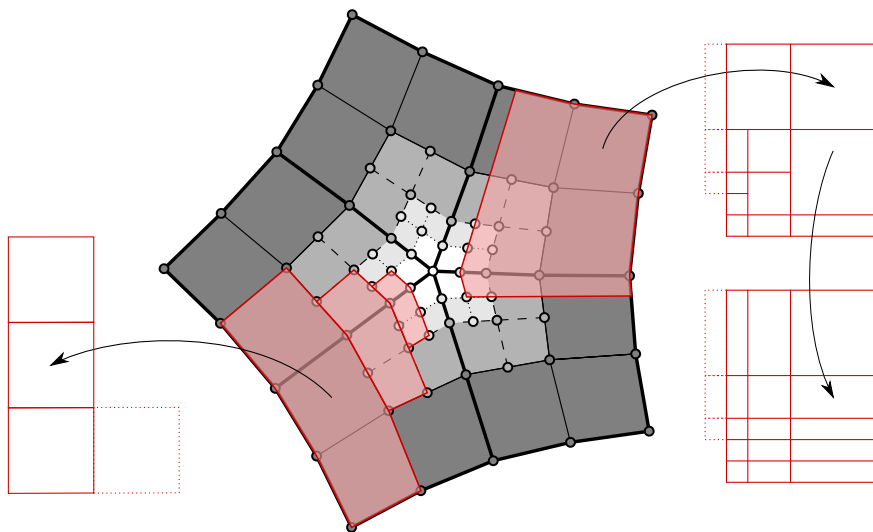
For completeness, we mention other approximating methods in general IgA context discussed in [MJ15; CST17], based on interpolation and look-up tables, and weighted quadrature per row of the element matrices, respectively.

### 4.2.2 Improving quadrature for Catmull–Clark elements (I)

Motivated by improved quadrature rules for IgA [HRS10; Aur+12] and optimal ones for splines [BC16a; BC16b; BC17], we joined forces with Michael Bartoň to derive quadratures for macropatches by means of *homotopy continuation*. In short, this can be interpreted as numerically tracing a curve from a known *source* quadrature rule to a *target* quadrature rule which is known to exist [MP77].

In the setting of subdivision splines and EVs, the idea is to derive quadrature rules for multiple tiles  $\Omega^{k,l}$  at once, in other words, quadrature for a macro patch. As Figure 4.6 shows, there are several options.

The *strip* macro patch consists of three tiles in a single ring, two in one element and the third one in a neighbouring element. In the case of subdivision splines, the

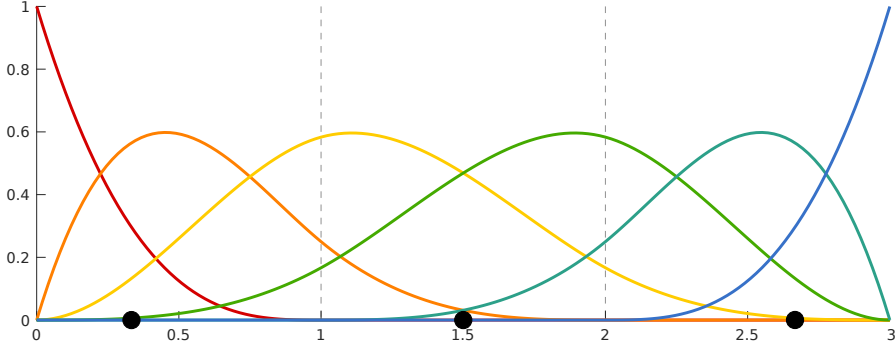


**Figure 4.6:** The five elements around an EV of valency  $n = 5$  collectively partitioned into rings. Various macro patches can be defined. At the bottom left we have three copies of the strip macro patch, which consists of three tiles in a single ring. At the top right a macro element is shown that covers the same area of the domain as the three strips together. Both macro elements are  $C^2$  in the interior. Illustration by Jiří Kosinka and Pieter Barendrecht.

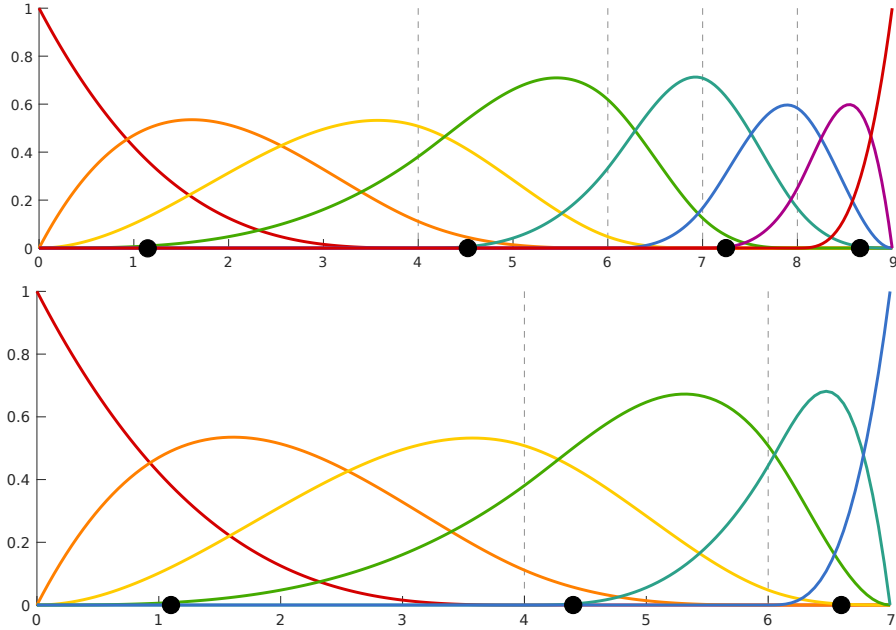
relevant space on this macro patch is  $(3, 2) \times (3, 2)$ , which denotes a space that is cubic and  $C^2$  continuous in both  $u$  and  $v$ . It turns out that the long side of the macro element requires 3 integration points; see Figure 4.7. The short side only covers a single tile, so it uses the default 2 Gauss–Legendre integration points.

Alternatively, macro patches with tiles from multiple levels can be considered. The idea is to consider a strip macro patch on a certain level  $r$ , and extend it back to level 1 such that it (partially) covers tiles from the previous levels. Before a quadrature rule can be determined for such a macro patch, the knot-lines from level  $r$  have to be extended back to level 1. This *virtual* knot-insertion is shown in Figure 4.6 on the right for the  $r = 3$  macro element. For the space of subdivision splines, in this case the homotopy continuation approach produces quadrature rules with 4 integration points for the long side and 3 points for the short side of the macro element. The total is therefore 12 integration points for the macro element – quite an improvement compared to the per-tile integration, which requires  $3 \cdot 3 \cdot (2 \times 2) = 36$  points to integrate the same region! Figure 4.8 illustrates the spaces.

Note that placing  $n$  copies of these macro elements cyclically around an EV covers all tiles up to level  $r$  (this includes the strip element, which actually corresponds to  $r = 1$ ). This is a very practical property – although larger patches could be considered by starting with strip elements of length 4 at level  $r$ , these would overlap when placing them around an EV in the same fashion. Once the quadrature rules for a macro patch are known, this property ensures that the actual integration can still be performed *per element*. For example, the two integration points in the third tile of the strip macro patch also appear rotated  $90^\circ$  in parameter space in the strip macro patch adjacent to



**Figure 4.7:** The  $(3, 2)$  space on the long side of the strip macro patch can be modelled using a knot-vector  $\Xi = [a, b, c, 0, 1, 2, 3, d, e, f]/2$ , with  $a, b, \dots, f$  any knots not violating the monotonicity property of  $\Xi$ . Although any such knot-vector would do in theory, we choose to use the open knot vector  $[0, 0, 0, 0, 1, 2, 3, 3, 3, 3]/2$  so that the resulting integration points lie in the interval  $[0, 3]$  and not outside it. The six resulting non-uniform B-splines are visualised. The number of required integration points (solid black dots) matches the expected minimum of 3, as together with the associated weights this provides 6 DoFs.



**Figure 4.8:** Similar to the strip macro patch explained in Figure 4.7, the  $(3, 2)$  spaces for the long and short sides of the  $r = 3$  macro element are modelled using knot-vectors  $\Xi = [0, 0, 0, 0, 4, 6, 7, 8, 9, 9, 9, 9]/8$  (top) and  $\Xi = [0, 0, 0, 0, 4, 6, 7, 7, 7, 7]/8$  (bottom), with expected minimum numbers of integration points 4 and 3, respectively.

it. In Figure 4.6 this is indicated with dashed outlines.

An interesting question is which  $r$  yields the most efficient quadrature rule, that is, requiring the fewest number of integration points. The result also depends on the spline space considered — although subdivision splines are a good test case, in practice we have to consider other spaces, including

- $(5, 1) \times (5, 1)$ . This space is considered for the computation of the surface area  $\mathcal{A}$  of a planar Catmull–Clark subdivision surface  $S$ :

$$\mathcal{A} = \iint_S 1 \, dS = \sum_k \iint_{\Omega} \det(\mathbf{J}) \, dudv = \sum_k \iint_{\Omega} \left( \frac{\partial x}{\partial u} \frac{\partial y}{\partial v} - \frac{\partial y}{\partial u} \frac{\partial x}{\partial v} \right) \, dudv.$$

- $(8, 1) \times (8, 1)$ . This space is considered for the computation of the volume  $\mathcal{V}$  enclosed by a Catmull–Clark surface  $S$ . Using the divergence theorem we saw earlier in this chapter, the volume integral can be converted to a surface integral. Choosing a vector  $(0, 0, z)$ , we have  $\iiint_V \nabla \cdot (0, 0, z) \, dV = \iiint_V 1 \, dV = \mathcal{V}$ . The equivalent right-hand side  $\iint_S (0, 0, z) \cdot \mathbf{m} \, dS$ , with  $\mathbf{m}$  the unit normal of the surface  $S$ , then turns into

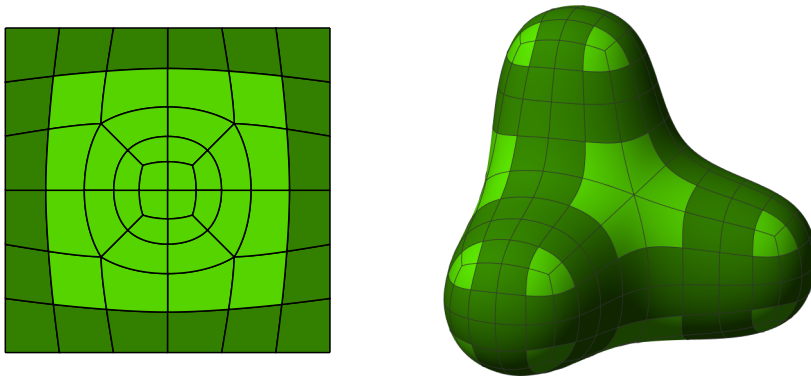
$$\begin{aligned} \mathcal{V} &= \iiint_V 1 \, dV = \iint_S (0, 0, z) \cdot \mathbf{m} \, dS = \sum_k \iint_{\Omega} (0, 0, z) \cdot \mathbf{m} \left| \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right| \, dudv \\ &= \sum_k \iint_{\Omega} (0, 0, z) \cdot \left( \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \right) \, dudv = \sum_k \iint_{\Omega} z \left( \frac{\partial x}{\partial u} \frac{\partial y}{\partial v} - \frac{\partial y}{\partial u} \frac{\partial x}{\partial v} \right) \, dudv. \end{aligned}$$

- $(4, 1) \times (6, 2)$  and  $(6, 2) \times (4, 1)$ . These spaces are considered for the computation of the stiffness matrix in FEA/IgA for Poisson’s equation. It follows from the integrand  $\nabla \varphi_{n,a} \cdot \nabla \varphi_{n,b}$ .

Comparison of different values of  $r$  for the various spaces (see [BBK18b] for full details) shows that in the case of both area and volume, the strip element ( $r = 1$ ) is the most efficient choice. For area, the space associated with the long side of the strip patch is of dimension 14, which leads to 7 integration points and -weights. For the short side we use the Gauss–Legendre rule with 3 points. For volume, the associated space for the long side is of dimension 23. As this is an odd number, no Gauss rule exists for it. Although a Gauss–Radau rule with 12 integration points exists, we choose to consider the superspace  $(9, 1)$  instead, which is of dimension 26. This results in 13 integration points and -weights. For the short side a Gauss–Legendre rule with 5 points suffices.

For both these applications, the *exact* values can be computed using the geometric series approach from Section 4.2.1, which is used to verify the newly derived quadrature rules. Note that the surface area of a 3D subdivision surface can *not* be computed exactly by this approach — the determinant of the Jacobian corresponds to the length of the cross-product of the two tangent vectors on the surface, which requires a square root. For planar surfaces, the cross-product always points in the (positive)  $z$ -direction, which means that the square root vanishes. In the case of volume, the determinant of the Jacobian vanishes because of the presence of the unit normal.

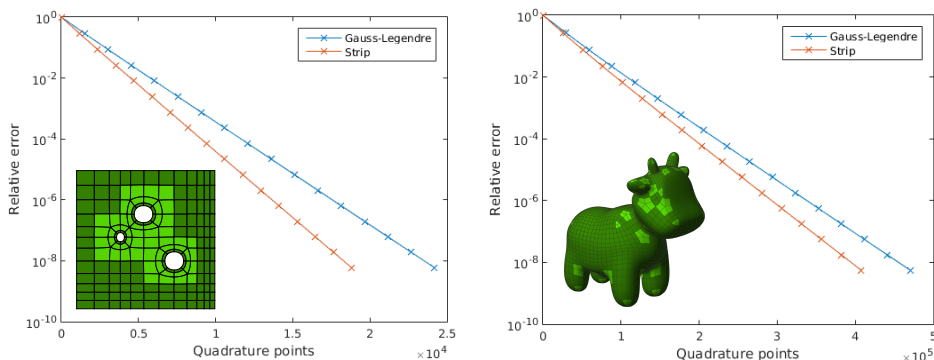
Also the results from the geometric series approach need to be verified. In the case of area, the benchmark is a mesh containing several EVs whose limit surface is the unit



**Figure 4.9:** Benchmarks for the exact area and volume computations. The planar subdivision surface (left) has an area of  $\mathcal{A} = 1$  by construction of the mesh. The subdivision surface associated with the tripod mesh (right) has a known volume of  $\mathcal{V} = 2.50400547615920543764371490988$ .

square (with a trivially known surface area). For volume, we use the *tripod* mesh whose volume is known [HR16]. See Figure 4.9.

Figure 4.10 shows two examples with a previously unknown surface area and volume, respectively. The use of the new quadratures for the strip element is compared to the per-tile use of Gauss–Legendre. For area, this comes down to the use of  $7 \cdot 3 = 21$  versus  $3 \cdot (3 \times 3) = 27$  points per level, and for volume  $13 \cdot 5 = 65$  versus  $3 \cdot (5 \times 5) = 75$  points per level. The improvement is less spectacular compared to the use of the  $r = 3$  macro patches to integrate the subdivision splines versus the per-tile approach in that case, but is still significant.

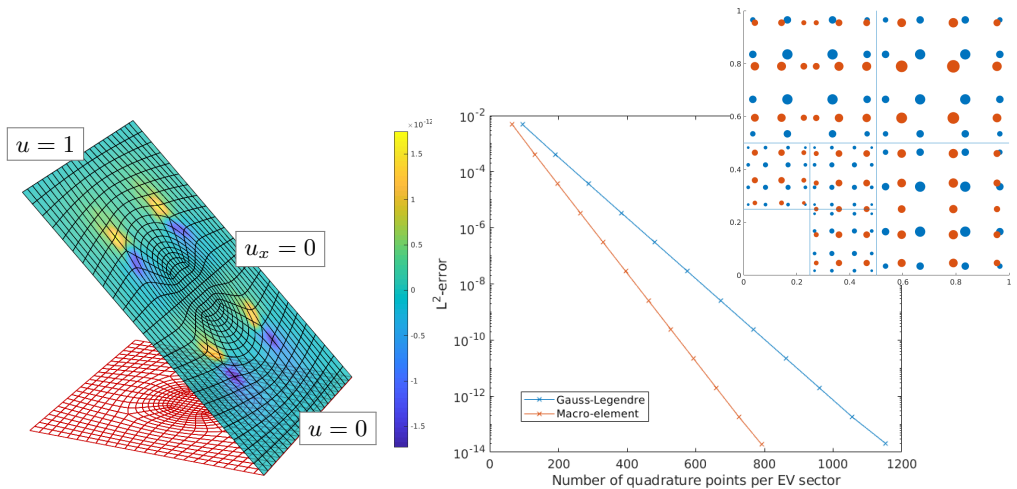


**Figure 4.10:** Relative errors when considering increasingly more levels of tiles around EVs to integrate the surface area or volume of the subdivision surfaces shown. The use of strip elements (orange) is compared to the use of per-tile quadrature (blue). The numerical errors for both approaches are identical up to machine precision. The total number of quadrature points only reflects the number of points used around EVs.

For the Poisson equation in the context of subdivision-based IgA, the macro element



with  $r = 2$  is the most efficient. However, matters are a bit more involved here. The required spaces  $(4, 1) \times (6, 2)$  and  $(6, 2) \times (4, 1)$  are not symmetric on the element. Using both of them would require  $(4 \cdot 10) + (6 \cdot 7) = 72$  integration points. If instead we consider the superspace  $(6, 1) \times (6, 1)$  containing both, the corresponding  $6 \cdot 11 = 66$  points suffice. Compared to the per tile-approach which needs  $2 \cdot 3 \cdot (4 \times 4) = 96$  points to integrate the same region, it is a considerable improvement. Another complication is the presence of inverses of Jacobians in the integrand (rational expressions for this planar setting) — as such, it is not possible to use the geometric series approach to obtain exact results. Instead, we use BVPs with known analytic solutions to verify the results. A straightforward example is Laplace's equation on a rectangular domain with Dirichlet boundary conditions on two opposite sides and homogeneous Neumann boundary conditions on the other two. The unique solution is linear — Figure 4.11 visualises the BVP.

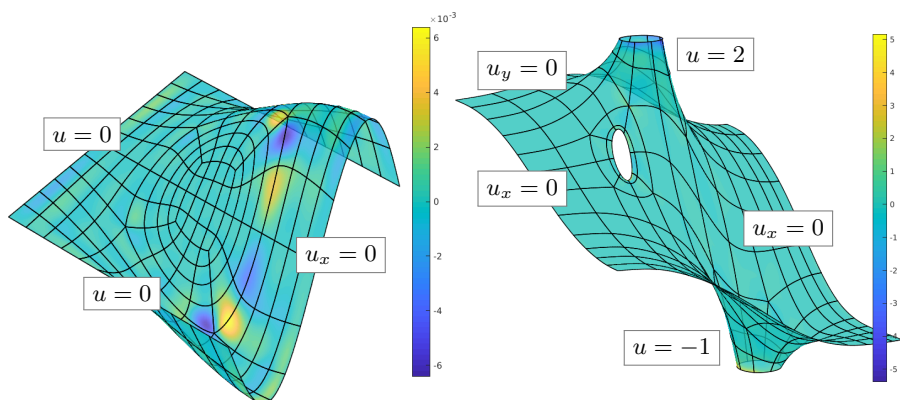


**Figure 4.11:** Solving the Laplace equation on a square mesh with the indicated boundary conditions imposed. The colour-coding (left) visualises the error (i.e. the difference between the analytic linear solution and the approximation). The two quadrature approaches — the  $r = 2$  macro patch (orange) and per-tile Gauss–Legendre (blue) are visualised (top-right) and compared using the  $L^2$  error of the approximation (right).

A more interesting example is Poisson's equation  $-\Delta u = \pi^2(5 \cos(\pi x) - 4) \sin(2\pi y)$  on the unit square, which has — using homogeneous Dirichlet boundary conditions on three edges and homogeneous Neumann boundary conditions on the remaining edge — the analytic solution  $u = \cos(\pi x) \sin(2\pi y) - \sin(2\pi y)$ . Figure 4.12 (left) shows the obtained approximation and its error.

Finally, we solve the Laplace equation on the planar domain with three holes we also used for the area computation. We impose Dirichlet boundary conditions on two of these holes, and use homogeneous Neumann boundary conditions elsewhere. As this BVP has no known analytical solution, we use a simplified residual to assess the quality of the approximation. The result is shown in Figure 4.12 (right).

Based on these three test cases illustrated in Figures 4.11 and 4.12, we can make the



**Figure 4.12:** Solving Poisson’s equation using the indicated boundary conditions. The colour-coding reflects the error (left). In addition, Laplace’s equation with the indicated boundary conditions is solved on a larger square with holes. The analytic solution is unknown — the colour-coding reflects the Laplacian of the approximation, i.e. a somewhat simplified residual (right).

following observations:

- The error for the Laplace BVP with a linear analytic solution is very low, namely  $\mathcal{O}(10^{-12})$ . This is expected, as the subdivision elements have linear precision (they are isoparametric elements after all). The boundary conditions are also imposed exactly — we choose to interpolate the corners of the given mesh instead of applying the default Catmull–Clark boundary stencil for corners, which results in the interpolation of the entire mesh boundary. This means that only the quadrature error is left. Although not the entire subdivision element is integrated (recall that only a finite number of levels is considered), we suspect that the error is actually due to the unbounded first-order partials of the eigenfunctions associated with the subdominant eigenvalues for  $n > 4$ , which is a result of using Stam’s parameterisation. This carries over to the partial derivatives of the subdivision splines, which likely causes numerical instabilities for the evaluation of the (inverse of) the Jacobians as well as the product of partials at integration points near the EV. This is backed up by the observation that the largest errors occur near EVs of valency  $n = 5$ , whereas regions near EVs of valency  $n = 3$  seem to behave considerably better, even though the approximation space on these elements is of a lower dimension than that of the  $n = 5$  elements. Unfortunately, the  $n = 3$  elements are also considerably smaller, which makes it difficult to be certain. Ultimately, it shows the importance of having  $\lambda_n = \frac{1}{2}$  for all valencies<sup>†</sup>, which is not the case for the Catmull–Clark scheme.
- The Poisson BVP gives us an idea of the behaviour of subdivision splines for a

<sup>†</sup>Recall that this results in a more uniform contraction upon subdivision (for Catmull–Clark,  $n = 3$  elements contract faster than their ordinary neighbourhood because  $\lambda_3 < \frac{1}{2}$ , whereas  $n = 5$  elements contract slower than their ordinary neighbourhood because  $\lambda_5 > \frac{1}{2}$ ). In addition, Stam’s parameterisation would be well-behaved with  $\lambda_n = \frac{1}{2}$  (i.e. bounded first-order partials).

solution that is not in the space  $V_h$ . The resulting error is considerable, and most pronounced around the regions with EVs of valency  $n = 5$ , most likely due to the reasons mentioned above.

- The Laplace BVP without known analytical solution highlights the behaviour at boundaries that are *not* interpolated and have a (Dirichlet) boundary condition imposed on them. In this case, we used *ghost points* [Sch96; LB07] to implement the boundary conditions, which effectively reduces the dimension of the space on those elements from 16 to 12. This clearly influences their approximation behaviour. It is a known issue, which could be resolved by using full Bézier edge conditions (see e.g. [Sab10]) or using modified subdivision weights [KSD14].

The overall conclusion is that the parameterisation of the subdivision splines as well as the eigenvalues of a subdivision scheme have considerable influence on the approximation errors. The former cause could be resolved considering reparameterisation on the characteristic map [Sta98a; BZ04; WP16]. The latter could be ameliorated by tuning the eigenvalues. Several such approaches are available, though are sparse in the context of subdivision-based IgA [ZSC18]. The bounded curvature tuning [ADS06; Sab+07] results in surfaces with improved behaviour around EVs in a geometrical sense, though because of the larger eigenvalues, its use in IgA would probably result in errors that are worse compared to classical Catmull–Clark. An interesting direction would be to consider the subdivision surfaces with *adjustable speed* [KP09; KP18] in the context of IgA.

Without new theoretical insights, we cannot improve quadrature for subdivision elements any further in a theoretical sense. However, numerically there is still a lot to gain. Section 4.2.4 considers our attempt in this direction.

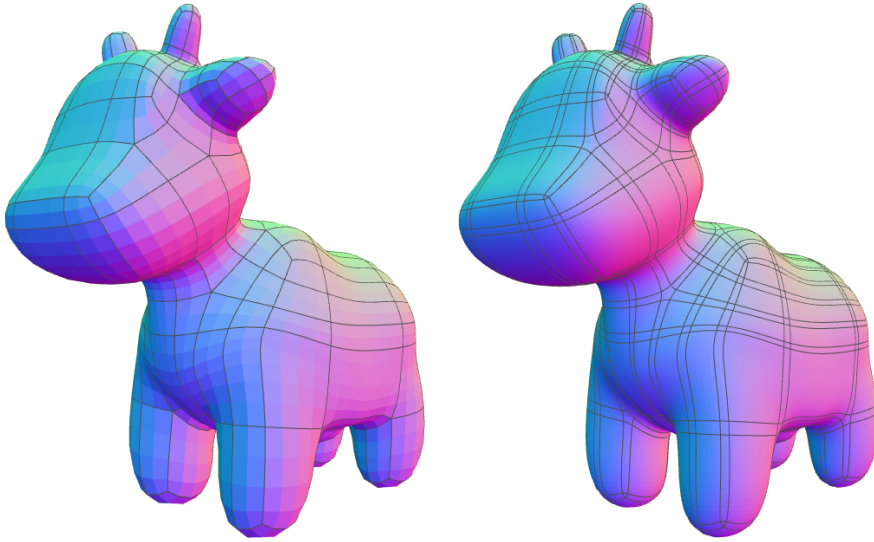
### 4.2.3 Patchification of the limit surface

Given the efforts taken to improve quadrature for subdivision elements (that is, those incident with EVs), it stands to reason to consider optimisation for the remaining parts of the surface, that is, the bicubic elements. In fact, and especially for repeatedly subdivided control nets, the regions around EVs only cover a small part of the entire surface. This motivated us to look into what we like to refer to as *patchification* of the limit surface.

The overall idea is rather straightforward. The first step is to consider the *separatrices* of the mesh, that is, the line strips that can be traced from EVs to other EVs (or the boundary of the mesh, if present). This naturally partitions the mesh into rectangular regions. However, these regions also include the faces containing the EVs, which correspond to the subdivision elements. As such, we offset the separatrices by one element, which results in *band separatrices*. Figure 4.13 compares the two approaches.

One approach is to take the resulting rectangular macro elements and — in the context of FEA/IgA — integrate them using quadrature rules that have been computed and tabulated beforehand. However, we could take it one step further by merging the rectangular regions into even bigger macro elements, which in general should result in the use of even fewer quadrature points<sup>†</sup>. This turns out to be quite a complex problem.

<sup>†</sup>Note that some macro patches could be periodic (see e.g. the legs and torso of the Spot mesh), which would allow the use of the *half-point* quadrature rule [HRS10].

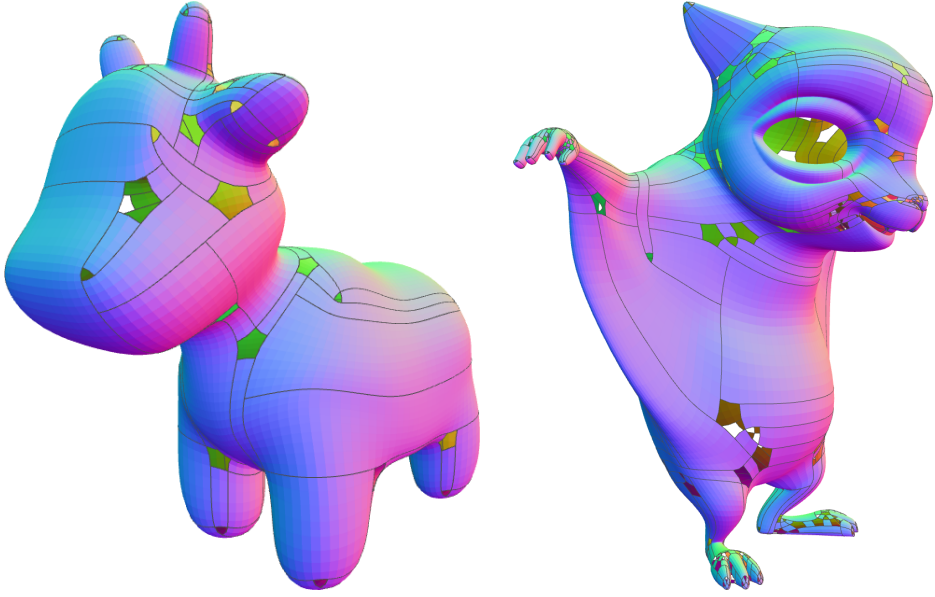


**Figure 4.13:** Separatrices for a subdivided Spot mesh (left) and band-separatrices for a Spot mesh that has been subdivided a bit more to highlight the resulting bands (right).

In addition, it is not immediately clear whether partitioning the surface in a minimal number of macro elements also results in the fewest number of quadrature points. A starting point is [Epp+08], which uses a so-called *motorcycle graph* to merge patches. It also includes a proof that the minimisation of patches is NP-complete. Other potentially relevant literature on *patchification* includes [CBK12; VY14; Cam17].

We conclude with a brief description of two ad-hoc approaches for merging patches without the aim of minimising their number. First, all band separatrices are constructed virtually (i.e. not actually drawn) and the number of intersections of each band separatrix with other band separatrices is determined and saved. Then for each corner of an  $n$ -sided hole around the EV (its one-ring neighbourhood), the band separatrix with the fewest intersections is selected. The first method then considers the starting points for these selected separatrices in random order, and draws them one-by-one. As soon as a band separatrix intersects another one, further generation is terminated. The second approach also considers the starting points in random order, but draws the separatrices one line piece at a time. For the meshes considered so far, the two approaches yield results that are largely similar. Two examples are shown in Figure 4.14.

Of course, it remains to be seen whether patchification is actually practical. In addition to possibly merging the macro elements resulting from drawing the band separatrices, the individual elements in a single macro element need to be indexed somehow such that they use the appropriate integration points. Additionally, it is not unlikely that there will be patches without associated integration points. Determining the quadrature rules should not pose a problem as they can be precomputed.



**Figure 4.14:** Merging rectangular macro patches into even bigger ones for a repeatedly subdivided Spot mesh (left) and for BLENDER’s Frankie mesh, which shows that the approach also works for meshes with boundaries (right).

#### 4.2.4 Improving quadrature for Catmull–Clark elements (II)

- The methods discussed in this section are work in progress and are expected to result in a future publication.

We now continue our efforts regarding the improvement of quadrature for subdivision splines. The reason to do so is that although the methods introduced in Section 4.2.2 improve the current default, the number of integration points required for a satisfactory result is still (prohibitively) large. This time, we take the road towards a numerical approach. That is, we generalise the approach considered in (4.20) to the subdivision splines. Recall that if all basis functions can be integrated exactly, then so can any function in their span.

Given the  $p = 2n + 8$  subdivision splines  $\varphi_{n,m}(u, v)$  and  $q$  unknown quadrature points  $(u_j, v_j)$  and -weights  $w_j$ , the following nonlinear system  $\Phi \mathbf{w} = \mathbf{b}$  should be satisfied for the quadrature rule to be exact (i.e.  $\mathbf{b}$  should be in the column space of  $\Phi$ ):

$$\begin{pmatrix} \varphi_{n,1}(u_1, v_1) & \dots & \varphi_{n,1}(u_q, v_q) \\ \vdots & & \vdots \\ \varphi_{n,p}(u_1, v_1) & \dots & \varphi_{n,p}(u_q, v_q) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_q \end{pmatrix} = \begin{pmatrix} \int \varphi_{n,1} \\ \vdots \\ \int \varphi_{n,p} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix}. \quad (4.35)$$

Note that in the literature such as system is often referred to as the *moment-fitting equations* [LJ75; BGR10; Hub+17]. In our setting, the exact integrals of the subdivision

splines (and alternatively of their derivatives and/or products) on the right-hand side can be computed using the geometric series approach. Alternatively, the integrals of the eigenfunctions could be considered instead of the subdivision splines.

We proceed to define a vector of residuals  $\mathbf{r} = \Phi \mathbf{w} - \mathbf{b}$ , with its individual entries defined as

$$r_m = \varphi_{n,m}(u_1, v_1)w_1 + \dots + \varphi_{n,m}(u_q, v_q)w_q - b_m. \quad (4.36)$$

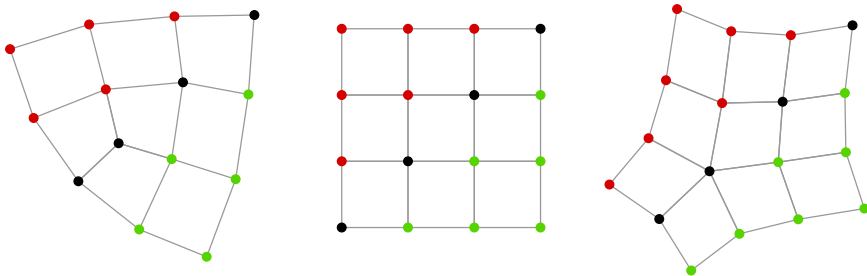
The aim is to get  $S = \mathbf{r}^T \mathbf{r} = \sum_m r_m^2 = 0$ , or in other words, the sum of squares of residuals should vanish, numerically spoken at least up to machine precision. We discuss two approaches that could achieve it.

The first method considers the use of the gradient of  $S$  with respect to all unknown  $(u_j, v_j, w_j)$  and uses an iterative approach to obtain  $\nabla S = \sum_m \nabla r_m^2 = \mathbf{0}$ . The partial derivatives with respect to the weights are  $\frac{\partial r_m^2}{\partial w_j} = 2r_m \frac{\partial r_m}{\partial w_j} = 2r_m \varphi_{n,m}(u_j, v_j)$ . Together, this can be expressed as

$$\nabla_w S = 2\mathbf{r}^T \mathbf{J}(w) = 2\mathbf{J}^T(w) \mathbf{r}, \quad (4.37)$$

with  $\mathbf{J}(w)$  the Jacobian matrix formed by the terms  $\frac{\partial r_m}{\partial w_j}$ . Clearly, we have  $\mathbf{J}(w) = \Phi$  in this case, so that  $\nabla_w S = 2\Phi^T(\Phi \mathbf{w} - \mathbf{b})$ . If the weights are the only unknowns, we have a linear least square problem, and the weights are obtained as  $\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{b}$ , which requires the columns of  $\Phi$  to be linearly independent.

However, solving only for the weights requires knowledge about the position of the integration points in the unit square  $\Omega$ . Unfortunately, this information is not available. The problem might be simplified somewhat by assuming a symmetric layout of the integration points (at least with respect to the diagonal  $u = v$ ), which could then be initialised randomly. Note that there is also a symmetry in the subdivision splines, which – together with the assumption of symmetric integration points – allows us to only consider  $(n+2)+4 = n+6$  of them; see Figure 4.15. The results obtained so far are *not* exact, and no patterns have yet been observed.



**Figure 4.15:** The subdivision splines associated with the black control points are symmetric with respect to the diagonal  $u = v$ . Furthermore, the subdivision splines associated with the red control points are reflections along  $u = v$  of those associated with the green control points.

Likewise, the partial derivatives with respect to the integration points gives  $\frac{\partial r_m^2}{\partial u_j} = 2r_m \frac{\partial r_m}{\partial u_j} = 2r_m w_j \frac{\partial \varphi_{n,m}(u_j, v_j)}{\partial u}$ . This results in

$$\nabla_u S = 2\mathbf{r}^T \begin{pmatrix} w_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_q \end{pmatrix} \frac{\partial \Phi}{\partial u} = 2\text{diag}(\mathbf{w}) \frac{\partial \Phi^T}{\partial u} \mathbf{r}, \quad (4.38)$$

mutatis mutandis for partials with respect to  $v_j$ . Clearly, the  $u_j$  appear only indirectly in (4.38), which means that they cannot be isolated and solved for directly. Instead, we have to resort to an iterative approach such as Gauss–Newton combined with a line search strategy, starting from a sensible initial guess. This is a nonlinear least squares problem.

Solving for only the integration points assumes the values of the weights to be known. Though this is not the case, we have at least some clues here — the weights should sum to one (to make sure that the constant function 1 is integrated exactly over  $\Omega$ ) and ideally be positive. Using random weights satisfying these constraints and using random initial guesses for the integration points often results in convergence such that (4.35) is satisfied close to machine precision, that is  $\mathcal{O}(10^{-13})$ . Note that this requires a solver supporting barriers or box constraints — the  $(u_j, v_j)$  are not allowed to leave  $\Omega$  during the iterations, as this would cause the subdivision level  $l$  to become Inf, negative or NaN<sup>†</sup>. So far, we have used the libraries NLOPT [Joh19] and ALGLIB [Boc19] for this purpose. The scatter plots shown in Figure 4.16 visualise the converged positions of the integration points over the unit square *without* enforcing symmetry on them. The results of 10000 converged runs are stacked, using transparency to highlight the density of the converged positions. We therefore refer to these plots as *density scatter plots*.

The resulting symmetric patterns are striking, and show intriguing clusters of integration points, the number of which appears to increase with the valency  $n$ . It should be noted that the individual solutions are generally not symmetric. So far, we have not been able to explain the occurrence and positions of these clusters. Note however that for  $n = 4$ , the clusters seem to ‘pin down’ the well-known  $2 \times 2$  Gauss–Legendre quadrature points.

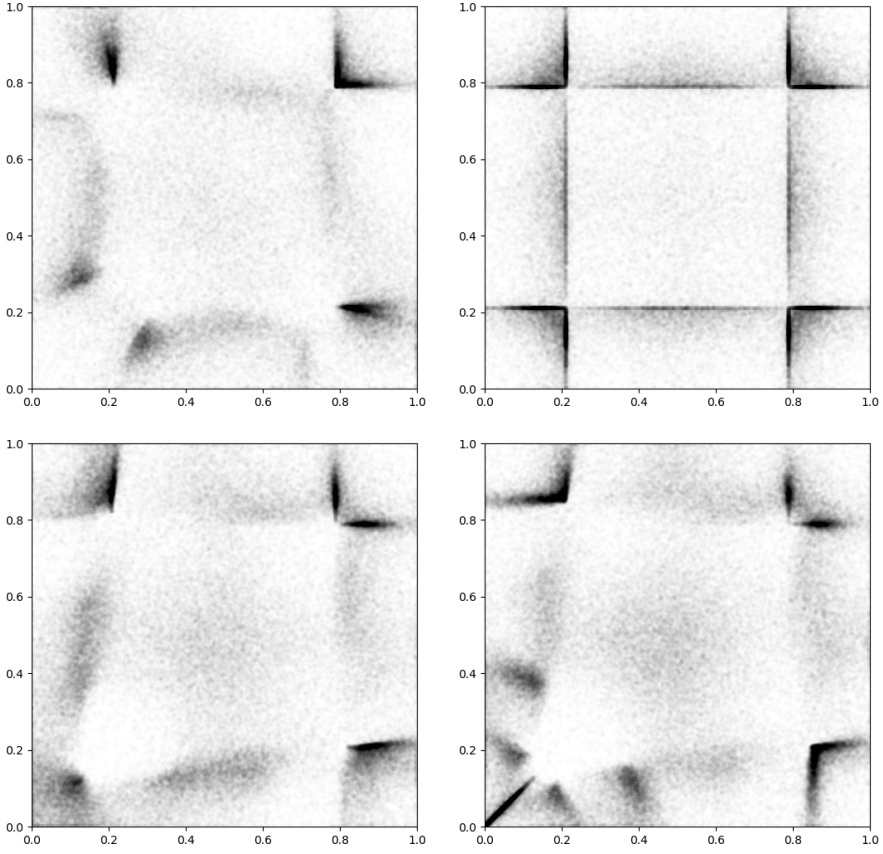
Another observation is that the higher the value of  $q$  is (i.e. the number of integration points) for a single valency  $n$ , the fuzzier the resulting density scatter plots are. This is shown for  $n = 3$  in Figure 4.17, using  $q \in \{7, 8, 9, 10\}$  integration points.

No convergence has been observed for  $q < n + 4$ , not even when running the code on a computational server for several days. This is not entirely unexpected — after all, there are  $2n + 8$  equations to be satisfied. Fixing random weights leaves  $2q$  DoFs, which most likely explains the behaviour. Still, solutions for lower  $q$  might exist. This is supported by the observation that the  $a \times b$  tensor-product Gauss–Legendre rule manages to integrate all  $4ab$  basis functions using only  $3ab$  DoFs.

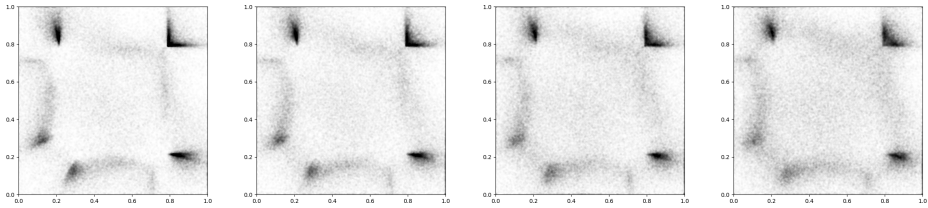
Considering the unknown integration points and -weights at the same time results in a mixed linear/nonlinear least squares problem. Effectively, the unknown triples

<sup>†</sup>Instead of the integration on a single element, we could consider the integration on the entire  $n$ -sided region around an EV at once, which should remove some of these constraints.





**Figure 4.16:** Density scatter plots visualising the converged positions of integration points overlapped for 10000 converged simulations using transparency. Top left:  $(n, q) = (3, 7)$ . Top right:  $(n, q) = (4, 8)$ . Bottom left:  $(n, q) = (5, 9)$ . Bottom right:  $(n, q) = (6, 10)$ .

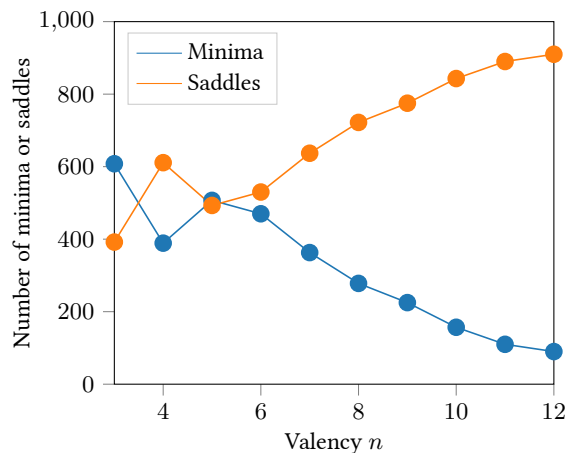


**Figure 4.17:** The patterns observed in 4.16 become fuzzier upon increasing  $q$ . Here, we show the results for  $n = 3$  for  $q \in \{7, 8, 9, 10\}$  visualised from left to right.

$(u_j, v_j, w_j)$  become unknown points in the unit cube. So far, we have not been able to obtain results. This might be due to the increased number of unknowns, or due to the different magnitudes of  $\nabla_w S$  compared to  $\nabla_u S$  and  $\nabla_v S$ . Another relevant insight is that the iterations get stuck at local minima or saddle points — the latter is more probable, as it is increasingly less likely to get stuck at a local minimum for an



increasing number of unknowns<sup>†</sup>. Figure 4.18 confirms this.



**Figure 4.18:** Number of (local) minima and number of saddle points encountered when running 1000 iterations for  $n = 3, \dots, 12$  using  $n + 4$  integration points based on the eigenvalues of the Hessian.

Another approach which so far has not proven fruitful is to iteratively alternate between optimising only for integration weights or for integration points — in practice the method quickly converges to a local minimum or saddle point.

Overall, this approach shows that subdivision splines *can* be integrated using a very small number of integration points. Though we have obtained many such rules (i.e. using  $n + 4$  quadrature points and -weights to integrate all subdivision splines of valency  $n$  *almost* exactly), we are still in pursuit of a symmetric quadrature rule. The importance of a robust solver supporting constraints was already touched upon, a point that is strengthened by the observation that for the *product* of (derivatives of) subdivision splines, the number of equations increases to  $(2n + 8)^2$ .

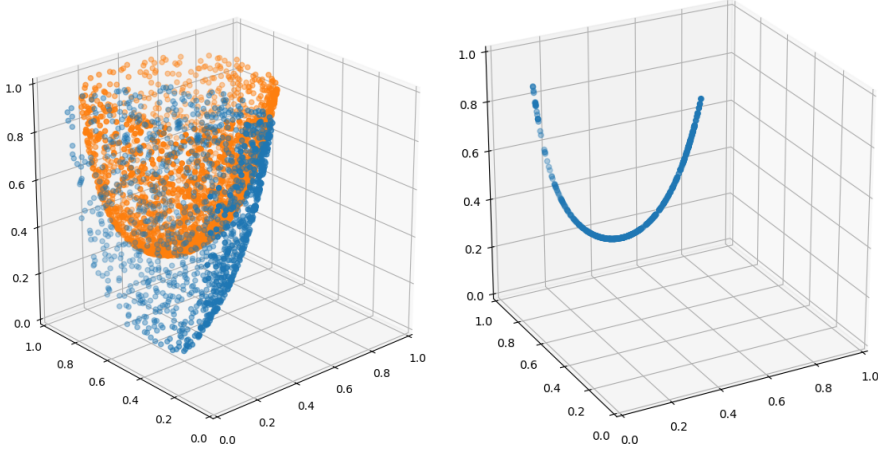
The second method is based on the observation that we do not merely want to have  $\nabla S = \mathbf{0}$  (which, as indicated above, could also result in convergence to a local minimum or a saddle point), but instead  $S = 0$ . I briefly worked on this with Michael Bartoň during my internship at BCAM, the Basque Centre for Applied Mathematics in Bilbao. The credits for the approach sketched below go to him.

Re-starting from the expression for an individual residual  $r_m$  (4.36), we can consider these as functions in  $3q$ -dimensional space. Setting  $r_m = 0$  then defines an isosurface in this space (generally a hypersurface). The intersection of  $p$  of these isosurfaces associated with  $r_1 = 0, \dots, r_p = 0$  then potentially gives us a  $3q - p$ -dimensional manifold of solutions on which all  $p$  residuals vanish. Unfortunately, the approach is difficult to visualise, unless we restrict ourselves to a single integration point  $(u_1, v_1, w_1)$  — this is of course not very practical, but allows a visual representation of the concept. As such,

<sup>†</sup>The logic here is that for high-dimensional spaces, the chances that the space locally curves ‘upward’ in all dimensions simultaneously — i.e. defining a local minimum — is not likely. It is more likely that the space locally curves ‘down’ in at least one direction, defining a saddle point.

consider the two isosurfaces in Figure 4.19 (which are actually associated with two bicubic Bernstein polynomials used for testing purposes). Their intersection results in a curve, that is, a  $3q - 2 = 1$ -dimensional manifold.

Note that the isosurfaces do not necessarily intersect (e.g. for certain choices for three isosurfaces associated with bicubic Bernstein polynomials, there is no mutual intersection – if there were one, it would be a single point), which results in the manifold being the empty set.



**Figure 4.19:** Two isosurfaces  $r_1 = 0$  (orange) and  $r_2 = 0$  (blue) visualised as point clouds (left). Their intersection is a curve (right), in other words, a one-dimensional manifold of solutions.

In the case of subdivision splines, we should consider  $2n + 8$  isosurfaces in a space of similar size – we could start with  $q = n + 4$ , as we know from the other approach that there are solutions in this case. The question is how to obtain a reasonable initial guess close enough to the manifold of solutions. Another idea is to start close to the intersection of some, but not all, isosurfaces, and trace this intersection until it intersects with another manifold, after the which the procedure can be repeated. However, we currently have no proof that these manifolds are guaranteed to be connected in  $[0, 1]^3$ .

As mentioned at the start of this section, the work presented here is research in progress and is expected to result in a future publication. Efficient (symmetric) quadrature rules and their performance in subdivision-based FEA/IgA (including the patch test) are eagerly awaited.

#### 4.2.5 Refinement for Catmull–Clark elements

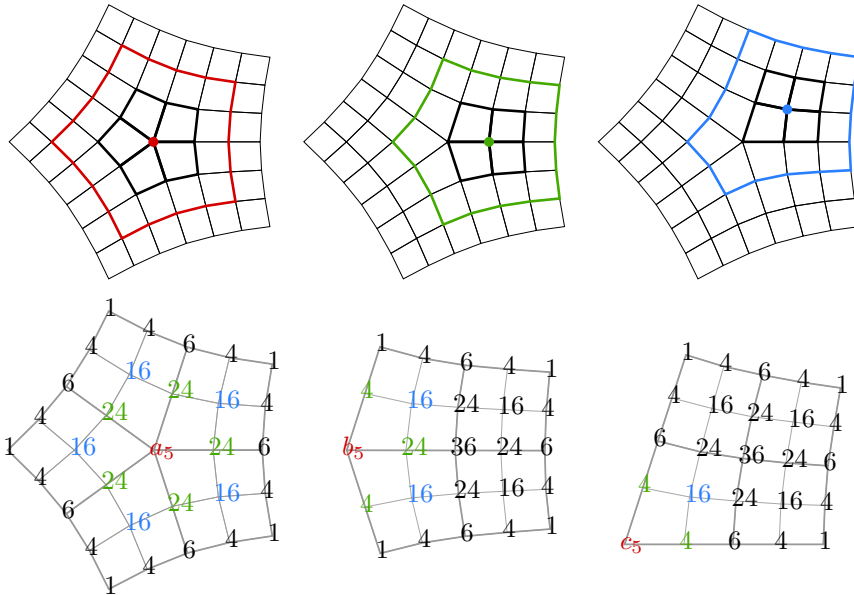
This exposition of subdivision-based IgA would not be complete without a note on the refinement possibilities of subdivision splines. Recall from the list of error sources in FEA that (local) refinement is crucial for enriching the space  $V_h$ , so that a better approximation can be obtained, resulting in a lower discretisation error.

Recall the *mask* of the uniform bicubic B-spline (i.e. the tensor-product of the mask of the uniform cubic B-spline  $[1, 4, 6, 4, 1]/8$  with itself), which shows how the bicubic

B-spline can be composed of scaled and shifted copies of itself. This was our starting point in the discussion of the Catmull–Clark scheme – the next step was to extract the *stencils* from the mask (i.e. the vertex, edge and face stencils) and generalise the former one to EVs.

In the setting of refinement, the following question needs to be answered – what are the masks associated with the subdivision splines  $\varphi_{n,m}$ ? To this end, let us consider a planar mesh consisting of an isolated EV of valency  $n = 5$  surrounded by three rings of faces, so that every sector consists of 9 faces. Next, we set the  $z$ -value of the central control point (i.e. the EV) to  $z = 1$ , whereas all other vertices stay at  $z = 0$ . The associated limit surface then corresponds to the subdivision spline associated with the EV. If we now apply one step of Catmull–Clark subdivision to the mesh and focus on the resulting  $z$ -values, it follows that together, these  $z$ -values *are* in fact the mask (recall that a step of subdivision does not change the limit surface associated with a mesh). The same approach can be applied to the vertices in the one-ring around the EV, which shows that in total there are only three different types of subdivision splines (plus the uniform bicubic B-spline), regardless of the valency  $n$ . Figure 4.20 summarises the above. Note that this approach also allows us to determine (or confirm) the support of a subdivision spline. Every subdivision step carries information from the central vertex one ring further, but every step the width of a ring is halved. As such, the support is the  $\sum_{k=0}^{\infty} \frac{1}{2^k} = 2$ -ring neighbourhood of a vertex.

4.2



**Figure 4.20:** The three types of subdivision splines for  $n = 5$  with their support outlined in red, green and blue (top). Applying one step of subdivision to the mesh defining a subdivision spline (i.e. a mesh with only the central vertex – indicated in red, green and blue – set to  $z = 1$ ) then reveals the mask (bottom). The non-symbolic coefficients should be normalised by a factor 64.

Given an object modelled as a subdivision surface, a global subdivision step replaces

all subdivision splines by the sum of subdivision splines on the refined grid which are weighted as dictated by the masks. The coefficients for each subdivision spline on the refined grid individually always add up to 1 — recall that this is the visual approach to define the stencils. For an example, consider Figure 4.20, where the subdivision spline on the refined grid associated with the EV receives the contributions  $a_5 + 5b_5 + 5c_5 = 1$ .

From the perspective of FEA/IgA, instead of using global  $h$ -refinement, local refinement is usually more efficient. Observe that the subdivision mask facilitates exactly this — it is perfectly fine to replace only a single subdivision spline by the appropriately weighted sum of subdivision splines on the refined grid. Generalising the approach, it is also possible to replace multiple subdivision splines simultaneously. In case this results in overlapping subdivision splines on the refined grid, their associated coefficients are simply added together. It can be interpreted as an extended version of hierarchical B-splines [FB88] (see also Appendix A), though seen from a different angle — using this approach, the refined basis still trivially partitions unity.

In the same fashion, the notion of *truncated* hierarchical B-splines [GJS12] can be extended. The idea is that certain subdivision splines on the refined grid would also contribute to subdivision splines on the original grid if these were replaced. As such, the refined subdivision splines can — using the right coefficients — be subtracted from the original coarse ones. In order to uphold the partition of unity, these coefficients should then be added to the coefficients associated with the refined subdivision splines.

Although matters are relatively straightforward when only two different levels of subdivision splines are considered, things quickly become more complex upon involving additional levels. In this scenario, it is not trivial whether the subdivision splines on the different levels are still globally linearly independent. Back in Cambridge in 2014, Urška Zore looked into this, using my existing MATLAB code as a starting point for the implementation. The resulting publication [ZJK16] extended the notions of [PW06]. Similar research was done in parallel by Xiaodong Wei et al., focusing more on the application of the above in the context of IgA. Their findings were published as two separate papers [Wei+15; Wei+16].

Summarising, using suitable refinement indicators, individual coarse subdivision splines can be replaced by weighted sums of refined subdivision splines, creating nested approximation spaces that allow for improved approximations  $u_h$ .

The other type of refinement in FEA known as  $p$ -refinement does not (yet) seem to have its counterpart in subdivision-based IgA. This would require the degree-elevation of the infinitely piecewise bicubic subdivision splines — all cubic  $C^2$  connections would then become quartic  $C^2$  connections. It poses an interesting direction for future research.

#### 4.2.6 The state of the art of subdivision-based IgA

We conclude this section with a summary of the state of the art of subdivision-based IgA. Given an object modelled as a (Catmull–Clark) subdivision surface, IgA takes away the geometry error that is often present in classical FEA. Next, using local refinement of the subdivision splines, the space  $V_h$  can be enriched so that the discretisation error can be made arbitrarily small (i.e. the method converges to the actual solution  $u$ ). Similarly, if desired, refinement at the boundary allows for better imposition of boundary

conditions, and further local refinement for better representation of the source term when projected onto  $V_h$ . Then, using one of our improved quadrature rules for subdivision splines, the quadrature error can be decreased significantly. Assuming a robust and accurate solver, the overall subdivision-based IgA process is then concluded to be more precise.

The discussion would not be complete without a pragmatic note on the behaviour of subdivision surfaces near their EVs. Although there are nowadays several approaches to improve this, the question is whether real-life objects exactly matching (Catmull–Clark) limit surfaces are actually desirable. Supported by the observation that these probably cannot be manufactured, the question becomes how much of an issue the approximation of the limit surface (e.g. using Gregory patches around EVs) would be?

## 4.3 The boundary element method (BEM)

The concepts of FEA as discussed above can be generalised to the trivariate case for running simulations on solids. The classical approach approximates the volume of interest using a tetrahedral mesh [ZTZ05; Si15], with the tetrahedra usually corresponding to linear elements, though higher-order ones also exist. An alternative is to use a hexahedral mesh, i.e. a mesh composed of brick elements. However, *hex-meshing* turns out to be considerably more challenging [Li+12].

Recall that in the bivariate case, IgA resolves the meshing challenge by directly using (parts of) the surface patches as elements. However, this typically does not extend to the trivariate case, as most objects nowadays are modelled as boundary representations (B-reps), lacking interior structure. As such, volumetric NURBS [ZL05], subdivision splines [Baj+02] or T-splines [Liu+14] need to be employed. Although this is certainly possible, these are all directions that are under active development. Trivariate NURBS are, like their bivariate counterparts, restricted to (piecewise) tensor-product geometries. Trimming might help, though is not straightforward. In the case of subdivision splines, there is currently no theory on the smoothness of *limit solids*. It stands to reason that the notion of the characteristic map can be extended, though note that in the case of subdivision solids, EVs propagate to the interior of the mesh as extraordinary edges (EEs), several of which might intersect.

Although each of these topics is an interesting research direction in its own right, we will not further elaborate on them, and instead briefly consider two alternatives for running simulations on volumes: the boundary element method (BEM) and the spline-enhanced finite element method.

The boundary element method can be an attractive alternative to FEA as (in general) only the boundary of the object of interest needs to be meshed. However, it does come with its limitations — the basic approach can only be applied to linear elliptic homogeneous PDEs (e.g. the Laplace and Helmholtz equations). Anything else that does not satisfy at least one of these criteria requires the use of either a specialised flavour of BEM, or can simply not be treated by BEM at all.

The principles of BEM are more mathematically involved compared to FEA. Many texts available on the topic place a lot of focus on these mathematical details without sketching a general approach first, which might be one of the reasons BEM is not ap-

plied in practice that much. In the remainder of this section, the aim is to give an overview of the approach known as *direct* (collocation) BEM.

### 4.3.1 The boundary integral representation for 2D Laplace

Our starting point is the Poisson BVP on a planar domain  $\Omega$  as discussed before in Section 4.1.1, that is,  $-\Delta u = f$ . Multiplying both sides by a function  $v$  (to be defined later) and subsequently integrating the result over the domain yields

$$-\int_{\Omega} v \Delta u \, d\Omega = \int_{\Omega} v f \, d\Omega. \quad (4.39)$$

Next, recall that due to the divergence theorem (4.5), the left-hand side can be expressed as

$$-\int_{\Omega} v \Delta u \, d\Omega = \int_{\Omega} \nabla v \cdot \nabla u \, d\Omega - \int_{\Gamma} v \frac{\partial u}{\partial n} \, d\Gamma. \quad (4.40)$$

The similarity between FEM and BEM ends here (at least temporarily). Applying the divergence theorem again with the roles of  $u$  and  $v$  reversed shows that the central term in (4.40) can be expressed as

$$\int_{\Omega} \nabla v \cdot \nabla u \, d\Omega = - \int_{\Omega} u \Delta v \, d\Omega + \int_{\Gamma} u \frac{\partial v}{\partial n} \, d\Gamma. \quad (4.41)$$

Plugging it back into (4.40) gives

$$-\int_{\Omega} v \Delta u \, d\Omega = - \int_{\Omega} u \Delta v \, d\Omega + \int_{\Gamma} \left( u \frac{\partial v}{\partial n} - v \frac{\partial u}{\partial n} \right) \, d\Gamma, \quad (4.42)$$

which is known as Green's second identity.

At this point, we introduce one of the key ingredients of BEM – the notion of a *fundamental solution* (also referred to as a free space Green's function) [Kat02; Bec92; Poz02]. In our case, we require the fundamental solution associated with the 2D Laplace equation, which is commonly derived within the context of potential theory. Without going into too much detail, consider a unit *point source* at a point  $P \in \mathbb{R}^2$ . The density at any *field point*  $Q \in \mathbb{R}^2$  due to this point source can be expressed as  $d(Q) = \delta(Q - P)$ , with  $\delta(\cdot)$  the Dirac delta function. We then look for the potential  $v$  such that  $-\Delta v = \delta(Q - P)$ . Using polar coordinates, it can be shown that [Bec92]

$$v = -\frac{1}{2\pi} \ln r = \frac{1}{2\pi} \ln \frac{1}{r}, \quad (4.43)$$

with  $r = |Q - P|$ , i.e. the Euclidean distance between  $Q$  and  $P$ . Using the property of the Dirac delta function that for any function  $g$  we have  $\int_{\Omega} g \delta(Q - P) \, d\Omega = g(P)$ , substituting the fundamental solution  $v$  into (4.42) then yields

$$-\int_{\Omega} v \Delta u \, d\Omega = u(P) + \int_{\Gamma} \left( u \frac{\partial v}{\partial n} - v \frac{\partial u}{\partial n} \right) \, d\Gamma. \quad (4.44)$$

Finally, we substitute (4.44) back into (4.39), which results in

$$u(P) + \int_{\Gamma} u \frac{\partial v}{\partial n} d\Gamma = \int_{\Gamma} v \frac{\partial u}{\partial n} d\Gamma + \int_{\Omega} v f d\Omega. \quad (4.45)$$

This shows that for a vanishing source term  $f$ , i.e. a homogeneous PDE, there are no more domain integrals. As a consequence, we no longer have to mesh the interior of the domain of interest, only its boundary. If  $f$  does *not* vanish, this is not the case, which negates most of the advantages of BEM. We note that there are several methods to get rid of this domain integral, with the dual reciprocity method [PB+12; GKW13] the most popular one. We will further assume  $f = 0$  and as such focus on BEM for the 2D Laplace equation.

Observe that (4.45) only holds for  $P \in \Omega$ . If  $P$  is not in the domain,  $-\int_{\Omega} u \Delta v d\Omega$  vanishes as in that case,  $\delta(Q - P)$  vanishes everywhere in  $\Omega$ . An important question is what happens when  $P$  lies on the boundary  $\Gamma$ . Intuitively, for a smooth boundary, the result should be  $\frac{1}{2}u(P)$ . Analysis shows that this is indeed the case [Bec92]. We can capture this using a coefficient  $c(P)$ , defined as  $c(P) = 1$  for  $P \in \Omega$ ,  $c(P) = \frac{1}{2}$  for  $P$  on  $\Gamma$  and  $c(P) = 0$  for  $P$  completely outside of  $\Omega$ .

Adopting the traditional notation  $q = \frac{\partial u}{\partial n}$ , we obtain

$$c(P)u(P) + \int_{\Gamma} u \frac{\partial v}{\partial n} d\Gamma = \int_{\Gamma} qv d\Gamma. \quad (4.46)$$

We can therefore focus on solving  $u$  and  $q$  on the boundary, after which  $u$  can be evaluated everywhere in the interior using the same expression<sup>†</sup>. Note that in the literature, the functions multiplying  $u$  and  $q$  in the boundary integrals in (4.46) are often referred to as *kernels*.

### 4.3.2 Discretisation

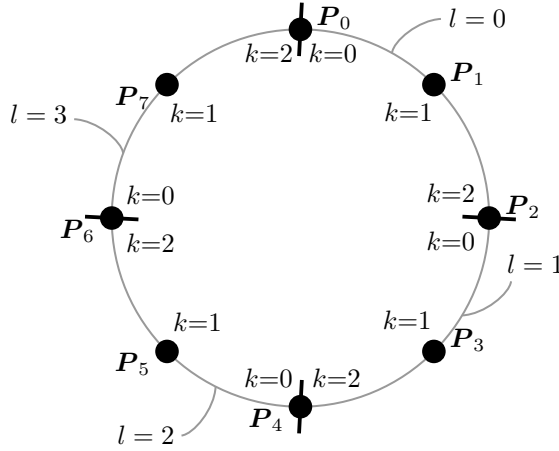
Our next step is to discretise (4.46). This is similar to FEA – the elements  $K_l$  can be parameterised using a reference element and a set of suitable local basis functions  $L_k(x, y) = R_k(F_l^{-1}(x, y))$ . The isoparametric principle can be invoked to represent the unknowns – plural. In BEM, both  $u$  and  $q$  are discretised (usually using the same set of basis functions, though two different bases can be used):

$$u_h(x, y)|_{K_l} = \sum_k u_k L_k(x, y), \quad q_h(x, y)|_{K_l} = \sum_k q_k L_k(x, y). \quad (4.47)$$

Meshing the boundary provides us with  $N$  nodes. Just as in FEA, each part of the boundary requires a Dirichlet, Neumann or mixed boundary condition, which translates to  $N$  nodal values for  $u$  or  $q$ . This leaves us with  $N$  unknowns. In FEA, the required equations naturally follow from the stiffness matrix. In BEM, a popular approach is to use *point collocation* [Poz02; GKW13], which places  $P$  at each node  $P_j$  consecutively. Note that this defines a different  $v$  and  $\frac{\partial v}{\partial n}$  for each node. Recall that  $Q$  is just shorthand notation for any point in  $\mathbb{R}^2$ . Integrating e.g.  $v$  over an element boundary  $\Gamma_l$  can therefore be interpreted as sliding  $Q$  along the element boundary.

<sup>†</sup>Note that this also allows us to consider *infinite* domains, which is very useful in the context of acoustics.

Assuming an interpolating basis (e.g. Lagrange elements),  $u(\mathbf{P}_j)$  corresponds to the nodal value  $u_j$ . Note that  $j$  is a *global* index, indexing all  $N$  nodes, whereas  $k$  is a *local* index, indexing the nodes within a single element  $l$ . An example is illustrated in Figure 4.21.



**Figure 4.21:** A BEM mesh composed of 4 quadratic Lagrange elements ( $l \in \{0, 1, 2, 3\}$ ) with black bars indicating the element boundaries. The nodes  $\mathbf{P}_j$  are indexed globally ( $j \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ ). Each individual element uses a local index ( $k \in \{0, 1, 2\}$ ) to refer to the appropriate nodes.

In the context of this example, placing  $P$  at  $\mathbf{P}_0$  results in

$$\begin{aligned} \frac{1}{2}u(\mathbf{P}_0) + \sum_{l=0}^3 \int_{\Gamma_l} u_h \frac{\partial v}{\partial n} d\Gamma &= \sum_{l=0}^3 \int_{\Gamma_l} q_h v d\Gamma \\ \Downarrow \\ \frac{1}{2}u_0 + \sum_{l=0}^3 \sum_{k=0}^2 u_k \int_{\Gamma_l} L_k \frac{\partial v}{\partial n} d\Gamma &= \sum_{l=0}^3 \sum_{k=0}^2 q_k \int_{\Gamma_l} L_k v d\Gamma. \end{aligned} \quad (4.48)$$

Focusing on the left-hand side, note that this can be interpreted as  $N = 8$  boundary integrals of (global) basis functions multiplied by  $\frac{\partial v}{\partial n}$ . The 4 integrals with basis functions associated with *shared* nodes consist of two contributions, whereas the 4 integrals of basis functions associated with *interior* nodes only have a single contribution (see Figure 4.21). Similar to FEA, we can compute the contributions per element, save them in a  $1 \times 3$  vector, and *assemble* the result into a  $1 \times 8$  vector. Repeating this with  $P$  moving to the next node  $\mathbf{P}_j$  until all nodes have been visited ultimately results in a  $8 \times 8$  matrix  $\mathbf{H}$ , where row  $j$  corresponds to  $P$  coinciding with  $\mathbf{P}_j$ . Similarly, the



right-hand side results in a  $8 \times 8$  matrix  $\mathbf{G}$ . Together, we obtain

$$\begin{pmatrix} H_{00} & H_{01} & H_{02} & H_{03} & H_{04} & H_{05} & H_{06} & H_{07} \\ H_{10} & H_{11} & H_{12} & H_{13} & H_{14} & H_{15} & H_{16} & H_{17} \\ H_{20} & H_{21} & H_{22} & H_{23} & H_{24} & H_{25} & H_{26} & H_{27} \\ H_{30} & H_{31} & H_{32} & H_{33} & H_{34} & H_{35} & H_{36} & H_{37} \\ H_{40} & H_{41} & H_{42} & H_{43} & H_{44} & H_{45} & H_{46} & H_{47} \\ H_{50} & H_{51} & H_{52} & H_{53} & H_{54} & H_{55} & H_{56} & H_{57} \\ H_{60} & H_{61} & H_{62} & H_{63} & H_{64} & H_{65} & H_{66} & H_{67} \\ H_{70} & H_{71} & H_{72} & H_{73} & H_{74} & H_{75} & H_{76} & H_{77} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{pmatrix} = \mathbf{G} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \\ q_7 \end{pmatrix}, \quad (4.49)$$

The  $1 \times 3$  element vectors are outlined for the first row of  $\mathbf{H}$  – the same pattern holds for the other rows as well. Additionally, note that the diagonal entries  $H_{jj}$  contain the extra contribution  $c(P) = \frac{1}{2}$ .

Once  $\mathbf{H}$  and  $\mathbf{G}$  are computed (details on the integration will be given in the next section) and assembled, the system  $\mathbf{H}\mathbf{u} = \mathbf{G}\mathbf{q}$  can be reorganised by collecting all unknowns in a vector  $\mathbf{b}$ , which then results in an expression of the form  $\mathbf{A}\mathbf{x} = \mathbf{b}$ . Note that unlike the stiffness matrix in FEA, both  $\mathbf{H}$  and  $\mathbf{G}$  are *dense* and generally *non-symmetric*. On the other hand, the BEM matrices are typically much smaller compared to  $\mathbf{K}$ , as only the boundary nodes are considered.

After obtaining the approximations of  $u$  and  $q$  on the boundary, the value of  $u$  can be evaluated anywhere in  $\Omega$  using (4.46). With all  $u_j$  and  $q_j$  known, this amounts to the evaluation of the boundary integrals

$$u(P) = \sum_l \sum_k \left( q_k \int_{\Gamma} L_k v \, d\Gamma - u_k \int_{\Gamma} L_k \frac{\partial v}{\partial n} \, d\Gamma \right). \quad (4.50)$$

### 4.3.3 Quadrature

The integrals in BEM, in our case those in (4.48), are generally more challenging to (numerically) compute compared to those appearing in FEM.

The integrals to compute the entries of  $\mathbf{G}$  contain the fundamental solution  $v$ . Observe that  $-\ln r = -\ln|P - Q|$  grows towards infinity when  $r$  approaches zero (i.e. when  $Q$  approaches  $P$ ), though the function itself is still integrable. For this reason, it is referred to as a *weak singularity*. It cannot be integrated accurately using Gauss–Legendre quadrature, but using *logarithmic quadrature* instead, it can [Poz02; Dav11]. The idea is to approximate an integral containing a logarithmic function and a function  $g(t)$  as  $\int_0^1 -\ln(t)g(t) \, dt = \int_0^1 \ln(1/t)g(t) \, dt \approx \sum_k g(t_k)w_k$ . The integration points  $t_k$  and -weights  $w_k$  can be obtained using the approach from (4.20). Using the cubic power basis as example again, we obtain

$$\begin{pmatrix} \int_0^1 \ln(1/t) \, dt \\ \int_0^1 \ln(1/t)t \, dt \\ \int_0^1 \ln(1/t)t^2 \, dt \\ \int_0^1 \ln(1/t)t^3 \, dt \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ t_1 & t_2 \\ t_1^2 & t_2^2 \\ t_1^3 & t_2^3 \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{4} \\ \frac{1}{9} \\ \frac{1}{16} \end{pmatrix}, \quad (4.51)$$

which yields the non-symmetric result

$$\begin{aligned}(t_1, w_1) &= (0.112008806166976, 0.718539319030385), \\(t_2, w_2) &= (0.602276908118738, 0.281460680969615).\end{aligned}$$

The use of these logarithmic quadrature rules is only required when  $P$  coincides with a node in the element we are integrating over *and* if the basis function currently under consideration does not vanish at  $P$ . Otherwise, Gauss–Legendre quadrature can be used. Although in the latter case a constant number of integration points can be chosen, it is often tuned based on the distance of  $P$  to the element of interest (the closer  $P$  is, the more integration points are used).

An alternative to the use of logarithmic quadrature is the use of *regularising transformations* which cause the (determinant of the) Jacobian to vanish at  $P$  [GKW13].

Talking about Jacobians, like FEA, we integrate everything on the reference elements. As such, the integrand also includes the determinant of the Jacobian to properly scale the line element  $dt$ .

The integrals involved in computing the entries of  $\mathbf{H}$  are unfortunately more problematic – these contain the normal derivative of the fundamental solution, which follows from repeated application of the chain rule:

$$\begin{aligned}r &= |Q - P| = \sqrt{(x - P_x)^2 + (y - P_y)^2}, \\ \frac{\partial r}{\partial x} &= \frac{1}{2} \frac{1}{\sqrt{(x - P_x)^2 + (y - P_y)^2}} 2(x - P_x) \\ &= \frac{1}{r} (x - P_x), \\ \frac{\partial r}{\partial n} &= \nabla r \cdot n = \frac{1}{r} \begin{pmatrix} x - P_x \\ y - P_y \end{pmatrix} \cdot n = \frac{1}{r} (Q - P) \cdot n, \\ \frac{\partial v}{\partial n} &= -\frac{1}{2\pi r} \frac{\partial r}{\partial n} = -\frac{1}{2\pi r^2} (Q - P) \cdot n = -\frac{(Q - P) \cdot n}{2\pi |Q - P|^2}.\end{aligned}$$

This shows that the integrals contain a factor  $\mathcal{O}(\frac{1}{r})$ , which is not integrable on a domain where  $r$  approaches zero. It is therefore referred to as a *strong singularity*. In general, this leads to the consideration of Cauchy principal values (CPV) of these integrals and methods based on those. However, in some cases – ours included – there is a straightforward approach to avoid these complications [Bec92; Dav11]. Note that if all  $u_j = 1$  (or some other constant), it follows that all  $q_j = 0$ . Physical examples (associated with BVPs very similar to ours) are a constant temperature ( $u$ ) which results in a zero flux ( $q$ ), or a rigid motion ( $u$ ) which results in zero tractions ( $q$ ). As the matrices  $\mathbf{H}$  and  $\mathbf{G}$  are not aware of the boundary conditions, we can temporarily assume these special ones, in which case  $\mathbf{H}\mathbf{u} = \mathbf{0}$ . This means that the rows of  $\mathbf{H}$  should sum to zero. As such,  $H_{jj} = -\sum_{i=0}^{j-1} H_{ji}$ . It follows that after all, we can use Gauss–Legendre quadrature for the computation of the entries of  $\mathbf{H}$ . The same remarks as those for the computation of the entries of  $\mathbf{G}$  apply.

Note that the integrand also contains the *unit* normal vector  $n$ . The (unnormalised) normal vector to the boundary curve can be obtained from the tangent vector to the

curve  $(\frac{dx}{dt}, \frac{dy}{dt})$  by rotating it  $90^\circ$ , i.e.  $(\frac{dy}{dt}, -\frac{dx}{dt})$ . The normalising factor to obtain the unit normal  $n$  is exactly the Jacobian, which means that these two factors cancel each other out<sup>†</sup>.

Finally, note that the integrals to evaluate  $u$  in the interior of  $\Omega$  (4.50) do not contain singularities, and can therefore be readily approximated using Gauss–Legendre quadrature (though care should be taken if the interior point of interest lies close to a boundary).

#### 4.3.4 BEM and IgA

Based on the number of recent publications, the introduction of IgA seems to have boosted research regarding BEM and has resulted in a combination of the two sometimes referred to as IgABEM. An important difference in IgABEM compared to classical BEM is that in the discretisation step the control points of the geometry are typically not interpolated. Like IgA, the nodal values therefore no longer represent physical values. This means that – when using point collocation – other collocation points have to be selected. Several options have been proposed in the literature [CHB09], including the images of parameter values where the basis functions attain their maximum value, as well as Greville abscissae (which coincide with the former in some cases).

The use of subdivision splines in the context of BEM was first considered in [BXW02] (using Loop’s scheme), which is around the same time subdivision splines were first used in FEA. Somewhat surprisingly, this publication has remained largely unnoticed, though it receives the due credits for pioneering the combination of these two topics. My introduction to subdivision-based BEM was the work of Timo van Opstal, who considered it as part of his PhD [Ops13], using my implementation of Catmull–Clark subdivision splines in the Python framework FINITY (which has since then been renamed to NUTILS). Results have been published in [OBZ15]. Nowadays there are multiple publications regarding subdivision-based BEM, including [Wan09; Ban+15], as well as related work focusing on T-splines [Sco+13; Gin+14].

There are many remaining aspects of BEM that have not been discussed here, including (local) refinement and optimisation. The former shares many characteristics of the approaches used in FEA. The latter covers a large area of research and includes techniques such as the *fast multipole method* [Liu09] and *hierarchical matrices* (H-matrices) [BGH03].

## 4.4 Spline-enhanced methods

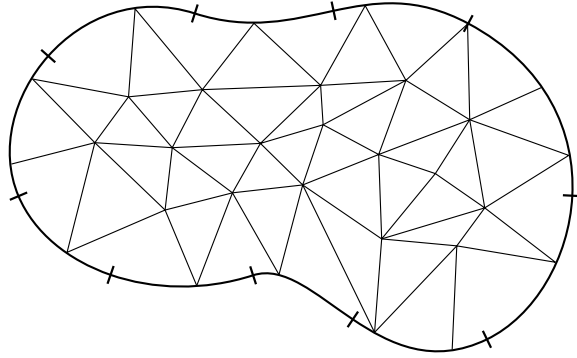
During my time in Leuven, Francesco Greco introduced me to the concept of NURBS-enhanced FEM (NEFEM). With subdivision-based IgA in mind, I proposed to look into subdivision-enhanced FEM. We collaborated on a paper, with the last bits completed during my second year in Groningen; it is published as [Gre+17]. Before we go into some details of subdivision-based FEM, the basics of 2D NURBS-enhanced FEM are laid out first.

---

<sup>†</sup>Recall that we have seen this happen before when considering the volume enclosed by a subdivision surface.

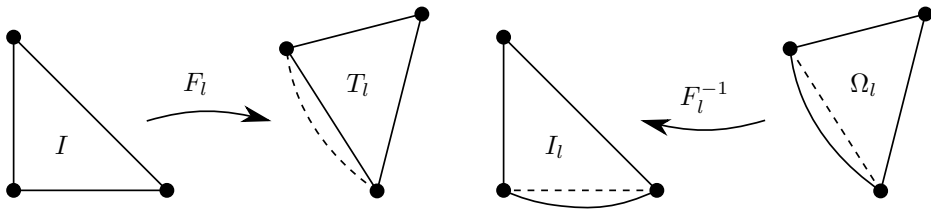
The ambitious goal of NEFEM [SFH08] is to get rid of two major shortcomings of classical isoparametric FEM. First, it aims at using the exact geometry of the object of interest. Additionally, it strives for proper polynomial approximation in the physical domain, regardless of the polynomial degree of the elements. This implies that the element maps  $F_l$  can be at most linear, resulting in constant Jacobians in the integrals required for computing the element matrices.

The exact representation of the geometry is achieved by first partitioning a given NURBS boundary into segments. Instead of restricting these segments to match the images of knot-intervals (resulting in points coinciding with the images of the knots), any partition can be used. Next, the resulting points on the boundary are used to triangulate the domain. This results in two types of elements — classical linear triangular elements in the interior, and augmented triangular elements at the boundary. A restriction here is that at most one side of an augmented element can coincide with the boundary. Figure 4.22 visualises the concept.



**Figure 4.22:** Cubic NURBS boundary (bars indicating the images of the knots) partitioned into segments that are subsequently used to triangulate the domain. The interior elements are classical linear elements, whereas those on the boundary are augmented elements.

Next, the observation is that since the corners of a reference element  $I$  can be mapped to the corners of any triangle  $T_l$  using a linear map  $F_l$ , the inverse of this map can be applied to the physical element  $\Omega_l$  to define a local curved element  $I_l$ . Figure 4.23 illustrates the maps.



**Figure 4.23:** A reference element  $I$  can be mapped to the triangle  $T_l$  using a linear map  $F_l$  (left). The inverse of this map can then be used to map the actual element  $\Omega_l$  back to the parametric domain, resulting in a local curved element  $I_l$  (right).

Note that as the isoparametric approach is *not* applied here, in principle any basis can be defined on  $I_l$  (or equivalently, on  $\Omega_l$ ). The approach taken in NEFEM is to define a Lagrange basis of degree  $d$  on the elements. Immediately, the question arises how the element nodes should be distributed. A simple approach is to place them all on  $I$ , so that a single reference element suffices. However, in this case the resulting Lagrange basis functions need to be considered (far) past the region containing the nodes<sup>†</sup>. As a consequence, the basis functions associated with interior nodes generally do not vanish on the curved boundary. This, in turn, results in issues regarding the strong imposition of Dirichlet boundary conditions<sup>‡</sup>. Alternatively, the element nodes can be (re-)distributed over each  $I_l$  individually. In this case, the resulting basis functions are still ‘extended’ past the region containing the nodes, though in a less extreme manner. Only when the boundary curve is (locally) captured by the Lagrange basis does this problem not occur. Details regarding node distributions mitigating the above can be found in the original publications [SFH08; Sev09; SFH11], though a clearer explanation is provided in [Ern11].

Next, we consider the numerical integration. Boundary integrals (i.e. the integrals incorporating the Neumann boundary conditions that naturally appear in the weak form) are handled as usual — in general the Jacobian is nonlinear here. The integrals over the augmented elements require a special quadrature. The original work [Sev09] considers different approaches, including a tensor-product Gauss–Legendre scheme (collapsing the  $v = 1$  edge into a point) and a high-order triangular quadrature rule. The Jacobian here is constant by construction.

Clearly, handling the augmented elements is more involved compared to the classical elements. However, as these augmented elements usually take up only a fraction of the total number of elements, it should not be a bottleneck. An advantage of the NEFEM mechanism is that (local) refinement is built-in — the boundary curve can be partitioned into a larger number of elements, or a higher-order Lagrange basis can be constructed on the elements. Naturally, both strategies also hold for the interior elements. A drawback is that NEFEM does not enjoy the nice properties of the NURBS (i.e. in contrast to IgA, their high-order continuity does not carry over to the approximation space), which makes it a  $C^0$  method. Nevertheless, for several settings NEFEM is shown to satisfy the theoretically optimal convergence rate, with more accuracy compared to classical FEA [Sev09; Ern11].

The extension of NEFEM to 3D largely follows the same principles. The NURBS surface is partitioned into triangular patches, which can again span multiple knot-intervals. The interior of the object of interest is meshed using tetrahedra. This results in two types of augmented elements — one which replaces a triangular face of a tetrahedron by a triangular patch, and another which merely replaces a straight edge of a tetrahedron by a curve.

Partitioning something that is inherently rectangular into triangles might appear somewhat unnatural. In theory, quadrilateral patches (aligned with the knot-lines,

<sup>†</sup>Whether this poses a problem or not depends on the context. Recall the Lagrange polynomials used to derive Gauss–Legendre quadrature, which do not have nodes at the boundaries of the parameter interval.

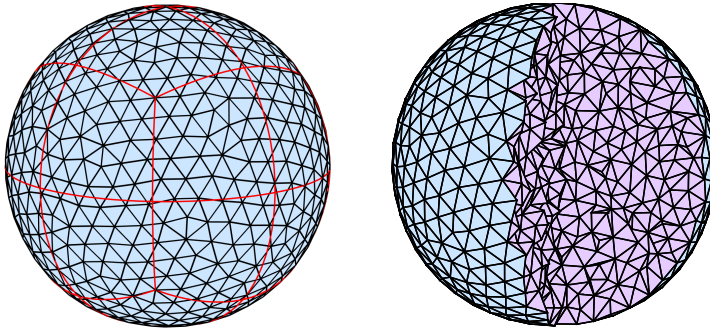
<sup>‡</sup>Strongly enforcing Dirichlet boundary conditions means to assign values to nodes on the Dirichlet boundary and subsequently remove the associated equations from the linear system  $\mathbf{K}\mathbf{u} = \mathbf{q}$ . Weakly imposing the boundary conditions can be done in various ways, including Lagrange multipliers, penalty methods and Nitsche’s method.

though not necessarily the knot-intervals) could be considered, connected to e.g. pyramid, prism or brick elements. However, volumetric meshing using these elements is considerably more challenging than constructing a tetrahedral mesh. Still, an augmented pyramid element would be compatible with a mesh that otherwise consists of tetrahedra, and would in my opinion have been a more natural choice.

#### 4.4.1 Subdivision-enhanced FEM

In this section, we summarise how to adapt the mechanics of 3D NEFEM to objects modelled as Catmull–Clark subdivision surfaces. Our initial approach (and in the end our only one) was to stay as close to NEFEM as possible. As such, we chose to partition the Catmull–Clark limit surface into triangular patches, which are used to define an interior tetrahedral mesh. In addition to the restriction that at most one face of a tetrahedron can coincide with the boundary, we require all EVs to be corners of augmented elements.

The construction of a (curved) triangle mesh on the limit surface can be approached in several ways. We selected a versatile approach that can generate both meshes of uniform and non-uniform density based on the iterative *force equilibrium* method described in [PS04a]. In our subdivision context, we use limit stencils to project EVs to their limit positions on the surface, as well as Stam’s method to evaluate the limit surface elsewhere. Following the triangulation, we chose TETGEN [Si15] to construct the interior mesh. An example is shown in Figure 4.24.

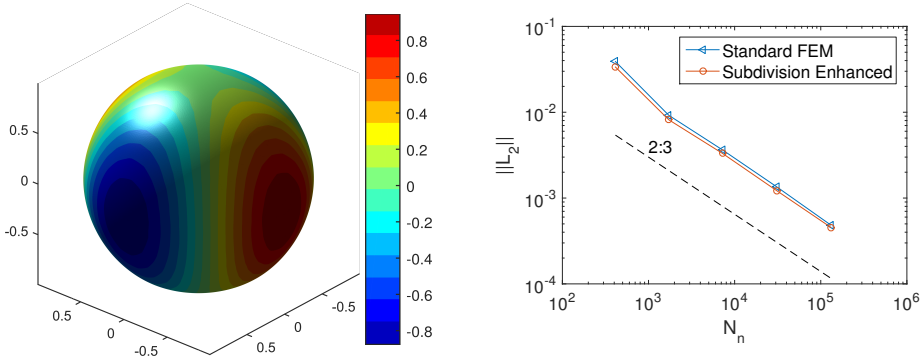


**Figure 4.24:** The Catmull–Clark limit surface associated with a once-subdivided cube partitioned into triangular patches (left). The projection of the control net is shown in red. Note that the EVs coincide with corners of the augmented elements. Based on this triangulation, the interior tetrahedral mesh is constructed (right), here shown from a different angle. Illustrations by Francesco Greco.

In order to keep things simple, we chose to use linear Lagrange polynomials as our physical basis functions, with the intention to generalise this later on. Like in NEFEM, ‘extensions’ of these basis functions have to be considered in the curved regions of the augmented elements. With regard to integration, the elements covering parts of multiple original subdivision surface patches are virtually split and integrated per part (following NEFEM). In addition, we also apply this virtual splitting procedure to the subpatches around EVs. Quadrature schemes were selected empirically. A patch test

was performed, once more confirming the need for quite a few levels of subpatches to reach machine precision.

The numerical examples include Laplace and Helmholtz BVPs, largely avoiding the use of Dirichlet boundary conditions for reasons stated above. Results for the Laplace BVP with Neumann boundary conditions corresponding to an analytical solution of  $x^2 - y^2$  are shown in Figure 4.25.



**Figure 4.25:** The approximation of the analytical solution  $x^2 - y^2$  (left). The  $L_2$ -norm of the error is plotted against the total number of nodes  $N_n$  (right), showing the expected convergence rate, as well as a slight improvement in accuracy compared to FEA. Simulations and illustrations by Francesco Greco.

Overall, our work should be seen as a proof-of-concept of extending NEFEM to subdivision surfaces. The improvements compared to FEA are marginal — combining this with the additional effort required for our method, it might not be the recommended choice. However, the consideration of higher-order bases on the elements, as well as improved quadrature schemes might yet increase our gain. Additional directions of future research include the extension to different subdivision schemes (e.g. Loop's scheme) and the use of pyramid elements parameterically aligned to Catmull–Clark surface patches (combined with classical tetrahedral elements for the interior of the object).



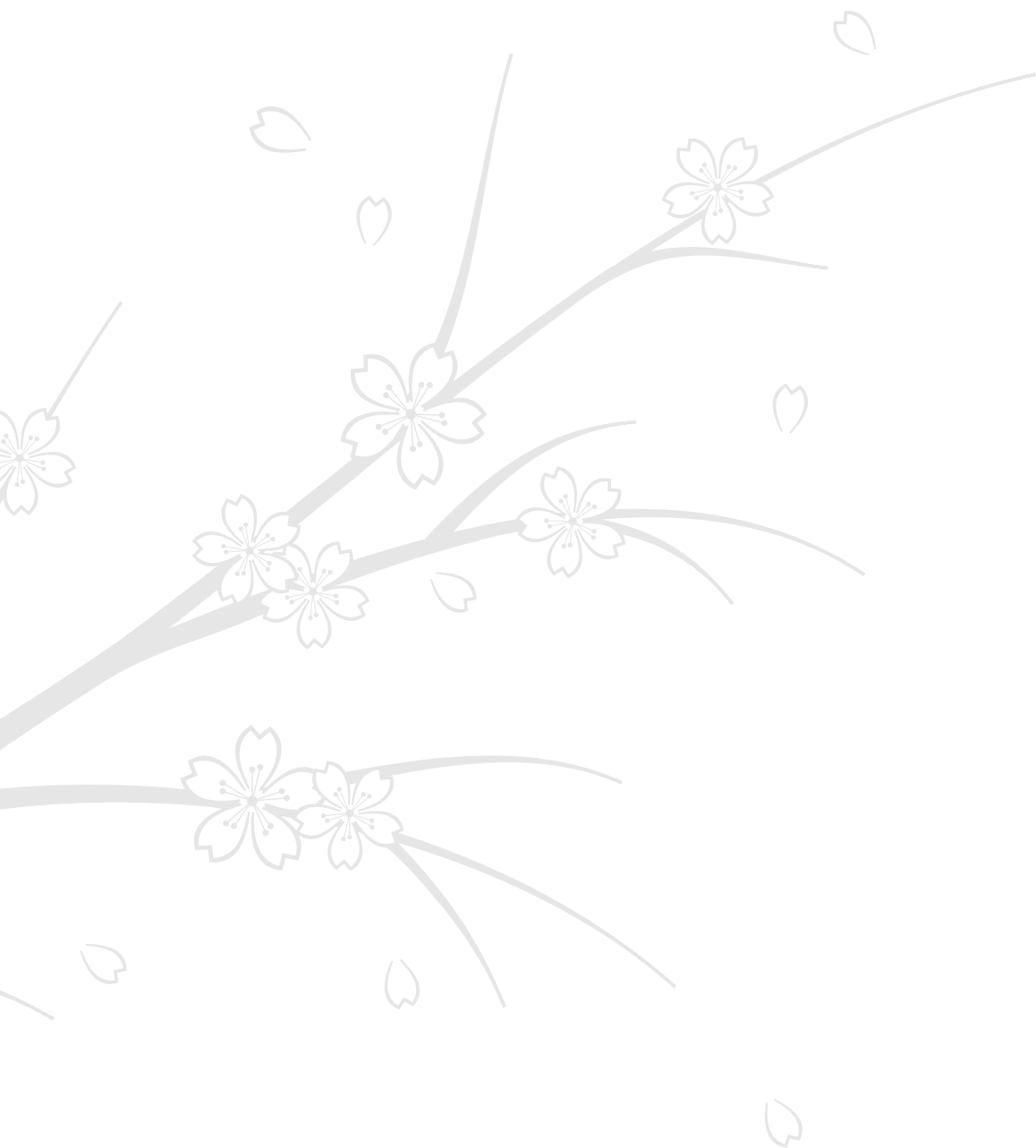


## 5 | Rendering aspects of curves and surfaces



Parts of this chapter have been published as

- Gerben J Hettinga, Pieter J Barendrecht and Jirí Kosinka. “A Comparison of GPU Tessellation Strategies for Multisided Patches.” In: *Eurographics (Short Papers)*. 2018, pp. 45–48. See Section [5.2.1](#).
- Jelle Bakker, Pieter J Barendrecht and Jirí Kosinka. “Smooth Blended Subdivision Shading.” In: *Eurographics (Short Papers)*. 2018, pp. 37–40. See Section [5.3](#).



In Chapter 2 we discussed univariate and bivariate splines from a mathematical point of view. In this chapter, we consider the visualisation of the resulting curves and surfaces. We discuss various available graphics APIs, look at implementation details and focus on two different applications — *hardware tessellation* (including approaches for multi-sided patches) and *subdivision shading*.

## 5.1 Graphics APIs

Over the last couple of years, the number of graphics APIs has grown. Although this might sound positive — after all, more to choose from — this might not necessarily be a good thing. Instead, a single, widely-supported open standard that is cross-platform is arguably preferable over a whole range of APIs.

For quite a few years, successive iterations of the Open Graphics Library (OpenGL) [KSS16; Wol18] provided exactly that. However, OpenGL has — for various reasons — become somewhat outdated, and has recently been succeeded by Vulkan [SK16]. Nevertheless, OpenGL remains the default choice for many projects, as it is accessible and relatively straightforward to work with. As such, this chapter will largely focus on OpenGL. A brief overview of alternatives (including Vulkan, Direct3D and Metal) is included for completeness.

### 5.1.1 OpenGL

In this chapter we consider *modern* OpenGL — that is, versions with a customisable pipeline composed of different *shaders*, altogether governed by *shader programs*. The primitives (i.e. building blocks) supported include points (`GL_POINTS`), lines (`GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`) and triangles (`GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`). In case of the latter two, the various options provide optimisations based on the connectivity of the lines and triangles. Coordinates and other attributes of vertices such as colours, normals and texture coordinates are typically uploaded to the GPU as a buffer object, commonly referred to as a vertex buffer object (VBO). A vertex array object (VAO) describes how the contents of a set of buffer objects should be interpreted (e.g. whether it contains 2D, 3D or 4D data<sup>†</sup>).

Initially, the data is usually described in *object* or in *world* coordinates. These have to be transformed to *camera* (or *eye*) coordinates, after which the *clipping* space, *normalised device* coordinates (NDC) and *screen* (or *window*) space follow. Internally, modern OpenGL is only aware of the latter three, and uses left-handed coordinates for these. The former three spaces are fully user-determined, though are traditionally taken to be right-handed.

The coordinate transformations are typically applied in the shaders, which are programmable bits of GLSL code. OpenGL recognises five different shader stages, all of which except the first are optional:

1. Vertex shader
2. Tessellation control shader (TCS)
3. Tessellation evaluation shader (TES)
4. Geometry shader
5. Fragment shader

---

<sup>†</sup>Examples include 2D and 3D geometry data, e.g. XY or XYZ, and 3D or 4D colour data, e.g. RGB or RGBA.

The typical role of the vertex shader is to transform coordinates using matrices, whereas that of the fragment shader is to assign colour to the fragments (which eventually make up the pixels). In between these two, the data is interpolated and rasterised.

Tessellation shaders (TCS and TES) introduce the `GL_PATCHES` primitive and are discussed in Section 5.2. Geometry shaders support the additional *adjacency* primitives `GL_LINES_ADJACENCY` and `GL_LINE_STRIP_ADJACENCY` for lines and in a similar fashion `GL_TRIANGLES_ADJACENCY` and `GL_TRIANGLE_STRIP_ADJACENCY` for triangles; they will be briefly discussed in the same section.

As OpenGL is language-independent, it can be used in combination with many different programming languages. In my PhD project, I have mostly used C++ when working with OpenGL. To create windows and an OpenGL context to render in, I chose to use the Qt framework — alternatives include combinations of GLFW and GLAD or GLEW, all of which are cross-platform.

### 5.1.2 Alternatives

Vulkan was already mentioned above as the successor of OpenGL. It is largely based on AMD's Mantle (development of which is now discontinued), and aims at higher performance compared to OpenGL. Methods to accomplish this include better distribution of the work to multiple CPU cores as well as multiple GPUs, and the use of shaders that have been translated to a binary format, SPIR-V<sup>†</sup>. It is much lower level than OpenGL, which gives the user more control over the hardware, though this also makes it considerably more complex to use. Although suitable for high-performance applications, without the use of wrapper- or convenience functions it might not be the best choice for prototypes and simple frameworks.

In addition to OpenGL and Vulkan, there are two other APIs that should be mentioned. Both are OS-specific (i.e. not cross-platform). On Microsoft Windows, Direct3D can be used (using shaders written in HLSL), whereas on macOS, there is Metal. Like Direct3D 12, Metal is largely based on the same concepts as Vulkan, though also covers general-purpose computing on GPUs (GPGPU), the general concept of which is briefly discussed below.

### 5.1.3 GPGPU

GPUs can be used for computations that are not directly related to graphics. In fact, because of their parallel nature, they are very well suited for all sorts of computations, collectively referred to as GPGPU. There are several frameworks through which this parallel processing power can be harnessed. The Open Computing Language (OpenCL) [Mun+11] is supported on a wide range of hardware and allows the user to implement and execute functions (referred to as *compute kernels*) on the GPU. Eventually, OpenCL will be merged with Vulkan. CUDA [SK10] is a powerful alternative to OpenCL, but can only be run on Nvidia graphics cards.

Although both OpenCL and CUDA can be combined with OpenGL using so-called interoperability functions, e.g. in cases where scene rendering involves heavy computation, OpenGL also offers the option of implementing a compute kernel as a *compute*

<sup>†</sup>OpenGL also supports SPIR-V as of version 4.6.

*shader*. Even though the possibilities of a compute shader are somewhat limited compared to what is offered by OpenCL and CUDA, it can be a convenient alternative because it can be used directly alongside the other shaders (though not within the same shader program) and is also programmed in GLSL. We note that also DirectX and Metal come with support for compute kernels.

When developing GPGPU code, the parallel computations can be thought of as a 3D grid. In terms of an OpenGL compute shader, a user-specified number of *work groups* is defined for all three dimensions — setting one or more of these to one effectively reduces the dimension of the grid to 2D or even 1D. Next, a subdivision of the work groups can be defined, referred to as the size of the work group. This size is again a vector in  $\mathbb{R}^3$  and is the same for all work groups.

Selecting the type of grid (3D, 2D or 1D) very much depends on the context of the problem. The subdivision of the work groups sometimes follows from the problem description — in other cases, tuning might be required to get optimal performance (which might also depend on the hardware used). As example, consider per-pixel ray-tracing, which calls for a 2D grid — one work group for each pixel. For each pixel, a number of rays or paths can be traced, which could be modelled using a 1D work group size. Because per-pixel ray-tracing can get very expensive (in particular in a real-time setting), another approach is to first triangulate the canvas, and only perform ray-tracing for the vertices of the resulting triangulation, which hints at the use of a 1D grid (again with a 1D work group size).

The common way to dispatch a computation using an OpenGL compute shader is to use

```
glDispatchCompute(nwgX, nwgY, nwgZ);
```

with `nwg{X,Y,Z}` the number of work groups in that dimension. Subsequently, the group size is defined in the compute shader itself as

```
layout (local_size_x = wgsX,
        local_size_y = wgsY,
        local_size_z = wgsZ) in;
```

with `wgs{X,Y,Z}` the work group size in that dimension. These dimensions are then available in the shader as `gl_WorkGroupSize`.

Alternatively, an OpenGL extension can be used to also define the work group size on the side of the CPU<sup>†</sup>:

```
glDispatchComputeGroupSize(nwgX, nwgY, nwgZ, wgsX, wgsY, wgsZ);
```

In the latest version of Qt at the time of writing, this command is not defined, but this can be solved using the following code snippet<sup>‡</sup>:

```
auto glDispatchComputeGroupSize = reinterpret_cast
    <PFNGLDISPATCHCOMPUTEGROUPSIZEARBPROC>(context()->
        getProcAddress("glDispatchComputeGroupSizeARB"));
```

In the shader, the following should be added:

<sup>†</sup>Somewhat surprisingly, this is a barely documented command, and therefore rarely used. For this reason, I have chosen to include it here.

<sup>‡</sup>Thanks to the Qt mailing list for helping me with this issue!

```
#extension GL_ARB_compute_variable_group_size : require
```

Next, the definition of the work group size is replaced by

```
layout (local_size_variable) in;
```

Unfortunately, this invalidates the use of `gl_WorkGroupSize` in the shader. Instead, `gl_LocalGroupSizeARB` should be used. Although this method is slightly more involved, it allows a developer or user to experiment with the group size without editing and recompiling the compute shader.

Input and output data for a compute shader can be stored as a shader storage buffer object (SSBO), which can be read from and written to in the compute shader. In a subsequent rendering call, this SSBO can then be used as a VBO. Alternatively, images (i.e. specific single-layer RGBA textures) can be used for loading and storing data, but are more limited. Other options for input data include texture buffers and uniforms.

Each item of a work group, often called *work item* or *invocation*, has its own *local* memory. In contrast, an SSBO or image is stored in *global* memory, as it should be accessible to all shader invocations. In addition to these, each work group has access to a limited amount of *shared* memory which is considerably faster than global memory. Shared data usually requires synchronisation, i.e. the use of memory barriers and atomic operations, details of which will not be discussed here.

In a CUDA setting, work groups are referred to as *thread blocks* and invocations as *threads*. The number of threads within a block that is executed in parallel on a multi-processor is called the *warp*, and has a *fixed* length of 32 (which follows from the core architecture of NVIDIA's graphics cards). It is therefore generally a good idea to have a thread block size that is a multiple of 32. In a similar context, for AMD's graphics cards this concept is referred to as the *wavefront*, and has a fixed length of 64.

#### 5.1.4 Web-based APIs

Given the amount of web-based content nowadays, it is surprising that the support for GPU-accelerated graphics on the web is still rather limited. Although the Web Graphics Library (WebGL) — which is based on OpenGL Embedded Systems (OpenGL ES) and uses a JavaScript API to draw on a HTML canvas element — offers basic functionality, it lacks support for features such as tessellation and compute shaders. It is, however, widely supported by web browsers. The same can unfortunately not be said of the Web Computing Language (WebCL), a JavaScript binding to OpenCL.

The prospect is that with WebGPU, a new standard based on Metal, Direct3D and Vulkan currently under development, more advanced functionality will become available in the near future. The current status of the project and its support by various web browsers can be checked at <https://github.com/gpuweb/gpuweb/wiki/Implementation-Status>.

## 5.2 Tessellation

Recall the Bézier curves introduced in Section 2.1.1. Given the control net of such a curve, how can we visualise the actual curve using OpenGL? Using only the vertex

and fragment shaders, de Casteljau's algorithm could be used on the CPU to sample the curve at e.g. equidistant parameter values or using a smarter strategy based on arc-length or curvature. The resulting points can then be uploaded to the GPU as a VBO and subsequently rendered using the `GL_LINE_STRIP` primitive.

A similar approach could be used to tessellate a triangular or square domain on the CPU for the evaluation of a triangular or tensor-product patch. `GL_TRIANGLE_STRIP` could be used to render the result (in the case of a triangular patch in combination with *primitive restart* as the triangle strips might be of different lengths), though connectivity of the vertices is less trivial than in the univariate case. A suitable alternative is to use index-based drawing, which stores each computed point only once (in contrast to the preceding method, which likely requires multiple copies of most point) and uses an additional index buffer object (IBO) to define the triangles.

OpenGL 3.2 (released in 2009) introduced the geometry shader stage, which allows the creation of (additional) geometry from within the shader. In theory, it facilitates the evaluation of quadratic or cubic Bézier curves on the GPU by using the `GL_LINES_ADJACENCY` primitive. Higher-order Béziers cannot be evaluated in this fashion — unlike `GL_LINES_ADJACENCY`, `GL_LINE_STRIP_ADJACENCY` does not provide simultaneous access to all control points.

Similarly, quadratic triangular Bézier patches could be evaluated on the GPU using the `GL_TRIANGLES_ADJACENCY` primitive. Unfortunately, there is no generalisation to (tensor-product) patches of higher degree — `GL_TRIANGLE_STRIP_ADJACENCY` does not allow concurrent access to the entire control net.

It should also be noted that in practice, using a geometry shader for (univariate) tessellation can be rather slow.

With the introduction of tessellation shaders in OpenGL 4.0 (released in 2010) and the `GL_PATCHES` primitive, matters simplified considerably. Given the unit parameter domain of a curve, triangular patch or tensor-product patch, the tessellation module can automatically tessellate it based on a sequence of user-defined tessellation levels provided in the TCS. In the TES, the parameter values (i.e. linear, barycentric or bilinear coordinates) corresponding to the points generated in the tessellated domain can be used to evaluate the curve or patch. In addition to integer levels (resulting in uniform tessellation of the parameter domain), *fractional* tessellation levels can be set which can be beneficial when interpolating between integer levels. It is also possible to manually re-map the generated parameter values, which can be useful for the crack-free tessellation of a composite surface containing T-sections.

The upper bound of the tessellation levels is limited, commonly with 64 as the highest level. Although certain tricks can be used to obtain a finer tessellation (e.g. the subsequent use of a geometry shader in the case of curves, or the use of texture buffers in the fragment shader in the case of patches with straight edges which allows pixel-level tessellation), this only works in specific cases.

In practice, various strategies of setting the tessellation levels can be used. The simplest is to use a uniform level throughout a scene. Alternatively, different tessellation levels can be used based on the distance of the curve or patch to the camera or to a contour or silhouette.

### 5.2.1 Tesellation for multi-sided patches

In Chapter 2 we mentioned the generalisation of Bézier triangles and tensor-product patches to multi-sided variants, though details were omitted. Hardware tessellation of such patches is not straightforward. In the following, we assume the use of generalised barycentric coordinates (GBCs) [Flo15] on regular polygonal domains  $\Omega_n$  with  $n$  indicating the valency of the polygon. That is, any point  $p \in \Omega_n$  can be expressed with respect to the vertices  $v_k$  defining  $\Omega_n$  using GBC functions  $\varphi_k$  as

$$p = \sum_{k=1}^n \varphi_k(p) v_k. \quad (5.1)$$

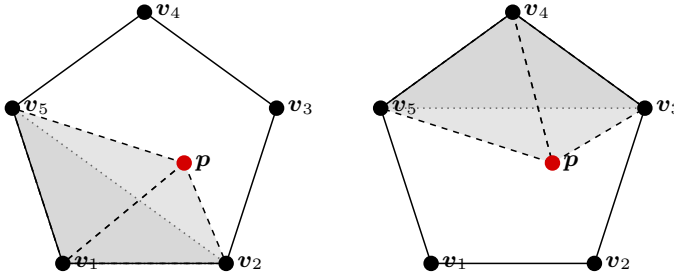
A common choice for  $\varphi_k$  are the Wachspress coordinates [Wac75], defined for convex polygons as

$$\varphi_k(p) = \frac{w_k(p)}{\sum_{l=1}^n w_l(p)}, \quad (5.2)$$

with

$$w_k(p) = \frac{\mathcal{A}(v_{k-1}, v_k, v_{k+1})}{\mathcal{A}(p, v_{k-1}, v_k) \mathcal{A}(p, v_k, v_{k+1})}, \quad (5.3)$$

where  $\mathcal{A}(\cdot)$  is the signed area of the triangle defined by the three arguments (using cyclic indexing)<sup>†</sup>, see Figure 5.1.



**Figure 5.1:** The triangles involved in computing  $w_1(p)$  (left) and  $w_4(p)$  (right) for the Wachspress coordinates for a point  $p$  (red) in the polygon  $\Omega_5$ .

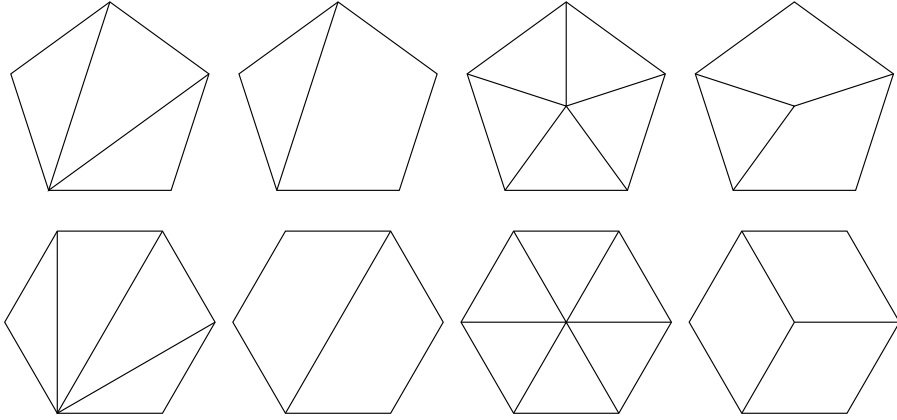
In addition to Wachspress, quite a few other GBC schemes are available, including *mean value coordinates* [Flo03] and *harmonic coordinates* [Jos+07].

The next step is to subdivide  $\Omega_n$  into triangular or quadrilateral sections, as these are the only bivariate domains supported by OpenGL's tessellation shaders. There are several approaches to this subdivision, as shown in Figure 5.2.

Depending on the choice of subdivision, we propose to use one of the following approaches:

<sup>†</sup>Note that very close to the boundary, the evaluation of this expression becomes unstable. Furthermore, on the boundary itself it cannot be evaluated, though as the GBCs are known to be piecewise linear here, this does not pose an issue.

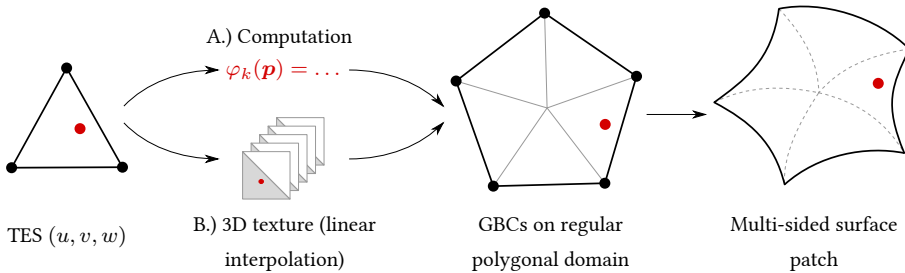




**Figure 5.2:** Various subdivisions of  $\Omega_5$  (top) and  $\Omega_6$  (bottom) into triangles and/or quadrilaterals. The latter two approaches on both rows require an additional point at the centroid of the domain.

- A.) The  $(u, v, w)$  or  $(u, v)$  coordinates in the TES for a triangular or quadrilateral patch define a point  $\mathbf{p}$ . Assuming  $\Omega_n$  to be inscribed in the unit circle, the (Wachspress) GBCs  $\varphi_k(\mathbf{p})$  can be computed with regard to the  $\mathbf{v}_k$  on the unit circle. If we limit ourselves to either all triangular patches (possible for all  $n \geq 3$ ) or quadrilateral patches (possible for even  $n \geq 4$ ), we can use OpenGL *instancing* to efficiently invoke the process multiple times.
- B.) In the case of centrally symmetrical subdivision, i.e. the third column in Figure 5.2, we could pre-compute the GBCs on a dense grid for one sector and store the result in a 3D texture (using one layer for each  $\varphi_k$ ). Note that this also allows the use of GBCs that are more expensive to compute, such as the Harmonic coordinates which are based on solving the Laplace equation.

The two approaches are summarised in Figure 5.3 for a centrally symmetric subdivision of  $\Omega_5$ .

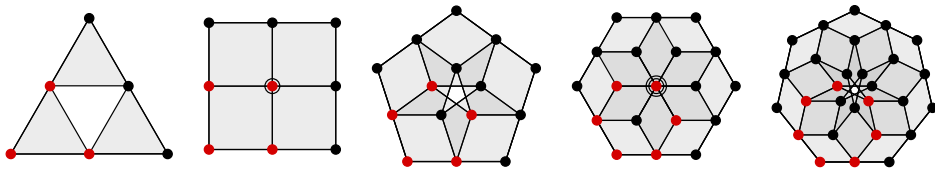


**Figure 5.3:** The two approaches to obtain GBCs for a point  $\mathbf{p}$  in a regular polygonal domain, which can then be used to render multi-sided surface patches.

Once we have the GBCs, we are in business and can render multi-sided surface

patches using OpenGL tessellation. Examples include extensions of Phong tessellation [BA08] and PN triangles [Vla+01] to general – possibly non-planar – polygons of arbitrary valency as described in [HK17].

In this context, it makes sense to include a discussion on a generalisation of triangular and tensor-product Bézier patches to  $n$ -sided patches known as *S-patches* [LD89; SS15]. Assuming regular polygons as parameter domains, the first step is to choose a degree  $d$  and uniformly subdivide the boundaries of the domain into  $d$  segments each. Then, just as in the triangular and quadrilateral setting, copies of the domain scaled by a factor  $\frac{1}{d}$  can be placed on top, indicating the positions of the control points. Note that the domain might not be completely tiled (e.g. in the triangular case there are gaps between the upward-pointing triangles) or that there might be overlaps between the copies (as is the case for  $n > 4$ ). Figure 5.4 shows the resulting control nets for  $d = 2$  for selected valencies.



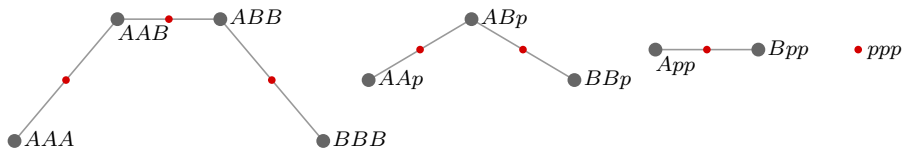
**Figure 5.4:** The control nets of quadratic *S-patches* for valencies  $n \in \{3, 4, 5, 6, 7\}$ . The control points generated by a single scaled copy are indicated in red. For even valencies, blossoming shows that several control points overlap, here indicated with concentric circles.

Next, de Casteljau’s algorithm is generalised to GBCs. Using blossoming, we compare the algorithm for a cubic Bézier curve, cubic Bézier triangle and a cubic pentagonal *S-patch*. To start with, we consider the univariate case for  $d = 3$ . Taking a general approach and assuming a parameter domain  $t \in [A, B]$ , any point  $p$  in this domain can be written as  $p = p_A A + p_B B$ , with  $p_{\square}$  linear functions of  $t$ . Recall that for  $A = 0$  and  $B = 1$ , these correspond to  $p_A = (1 - t)$  and  $p_B = t$ . In other words, assuming a blossom  $f(\cdot, \cdot, \cdot)$ , we have

$$\begin{aligned} & p_A f(A, A, A) + p_B f(A, A, B) \\ &= f(A, A, p_A A + p_B B) \\ &= f(A, A, p), \end{aligned}$$

which uses the *multi-affine property* of  $f$ . Labeling the control points with blossom values – using the shorthand notation  $AAA = f(A, A, A)$  – we can apply the approach to any pair of blossom labels differing in only one argument (the order of which is irrelevant because of the *symmetry property* of  $f$ ). This results in the red points shown in Figure 5.5. Finally, following de Casteljau’s algorithm, we repeat this until we are left with a single point corresponding to the blossom label  $ppp = f(p, p, p)$ , which by the *diagonal property* of  $f$  shows that it is the image of the point  $p$  (i.e. our curve evaluated at  $t = p$ ).

Moving on to a cubic triangular Bézier patch defined on the domain  $\triangle ABC$ , we can describe any point  $q$  in the domain as  $q = q_A A + q_B B + q_C C$ , where each  $q_{\square}$

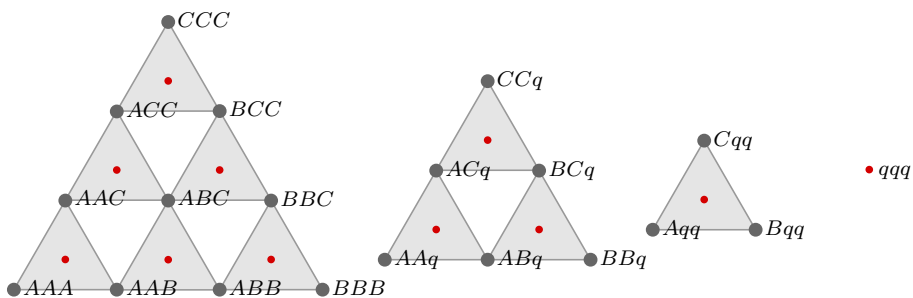


**Figure 5.5:** De Casteljau's algorithm for a cubic Bézier curve using blossoming notation. The image of a point  $p = p_A A + p_B B$  in the parameter domain  $[A, B]$  is computed in three steps.

corresponds to a barycentric coordinate  $\{u, v, w\}$ . We follow the same approach, using a blossom  $g(\cdot, \cdot, \cdot)$ , and obtain

$$\begin{aligned} & q_A g(A, A, A) + q_B g(A, A, B) + q_C g(A, A, C) \\ &= g(A, A, q_A A + q_B B + q_C C) \\ &= g(A, A, q). \end{aligned}$$

The point on the triangular surface associated with the point  $q$  in the domain again follows from de Casteljau's algorithm as illustrated in Figure 5.6.



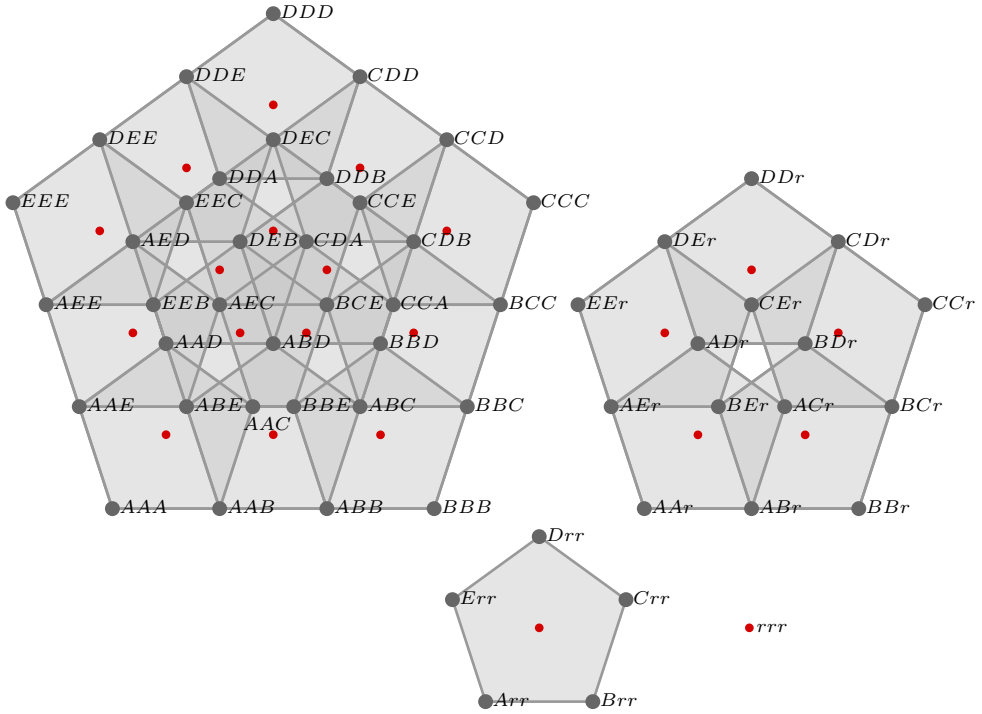
**Figure 5.6:** De Casteljau's algorithm for a cubic triangular Bézier patch using blossoming notation. The image of a point  $q = q_A A + q_B B + q_C C$  in the parameter domain  $\triangle ABC$  is computed in three steps.

Finally, we consider the cubic pentagonal S-patch. The elegance of the generalisation to S-patches is that we can use exactly the same approach as for the cubic Bézier curve and cubic triangular Bézier patch. Expressing a point  $r \in \Omega_5$  as  $r = r_A A + r_B B + r_C C + r_D D + r_E E$ , with the  $r_\square$  some generalised GBCs  $\varphi$ , we use a blossom  $h(\cdot, \cdot, \cdot)$ :

$$\begin{aligned} & r_A h(A, A, A) + r_B h(A, A, B) + r_C h(A, A, C) + r_D h(A, A, D) + r_E h(A, A, E) \\ &= h(A, A, r_A A + r_B B + r_C C + r_D D + r_E E) \\ &= h(A, A, r). \end{aligned}$$

After three iterations of de Casteljau's algorithm, we obtain the point of interest on our patch — see Figure 5.7.

This comparison shows that the definition and evaluation of S-patches is a natural extension of Bézier curves and Bézier patches. Although the layout of the control net



**Figure 5.7:** De Casteljau's algorithm for a cubic pentagonal  $S$ -patch using blossoming notation. The image of a point  $r = r_A A + r_B B + r_C C + r_D D + r_E E$  in the parameter domain  $\Omega_5$  is computed in three steps. Note that the domain (top left) is tiled with two layers of pentagons — the outer layer consists of 10 pentagons following the subdivision of the boundary for  $d = 3$ , whereas the inner layer consists of 5 pentagons.

is perhaps less structured for higher valencies and higher degrees compared to the triangular and quadrilateral equivalents, it can always be constructed based on the structure of the patch of one degree lower (and optionally verified by checking the blossom labels).

To conclude, we mention that there are other multi-sided surface patches, including transfinite surface interpolation over  $n$ -sided domains [VRS11] and generalisations of Gregory patches [HK18].

## 5.3 Subdivision shading

Now that we know how to properly tessellate surface patches, the final aspect to consider is the *shading* of these patches. Recall that, from a computer graphics point of view, the patches correspond to triangle meshes, even in the case of pixel-accurate rendering. Common approaches for shading polygonal meshes include

- Flat shading, which performs light computations using e.g. Phong's reflection

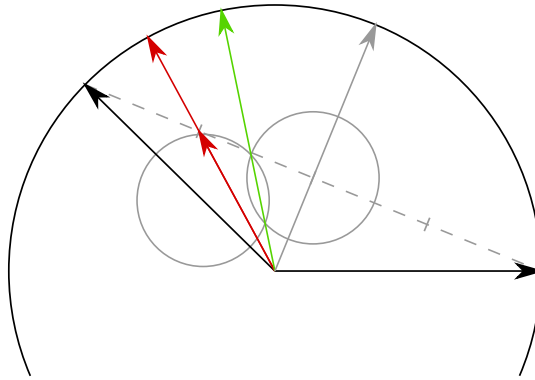
model at a single point on the polygon. The result is then used for the entire polygon.

- *Gouraud* shading, which performs light computations at the vertices of the polygon. The results are then interpolated over the polygon.
- *Phong* shading, which interpolates the normal vectors over the polygon and subsequently performs light computations per fragment.

Naturally, Phong shading is more computationally expensive compared to Gouraud shading, although on today's GPUs this is hardly an issue any more. Flat shading is used when it is important to distinguish individual polygons or for artistic purposes.

Regarding the interpolation of normals, consider the linear interpolation of two normalised vectors as shown in Figure 5.8. First of all, the resulting vector is typically not of unit length, and therefore needs to be normalised. Furthermore, the direction of the resulting vector might be different from what is expected.

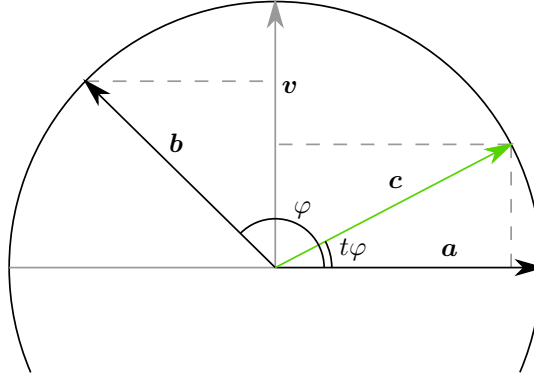
An alternative perspective considers unit normals as points on the unit circle. In that setting, taking the convex combination with e.g. coefficients  $\frac{3}{4}$  and  $\frac{1}{4}$  as in Figure 5.8 results in a vector that logically still has unit length and points into a (slightly) different direction. This *spherical linear interpolation*, commonly abbreviated as *slerp*, arguably yields more natural results.



**Figure 5.8:** Interpolation of two normalised vectors (black) using coefficients  $\frac{3}{4}$  and  $\frac{1}{4}$ , resulting in the red vector which is then normalised. Alternatively, *slerp* (here using a geometric construction involving the grey vector and two grey circles) can be used to obtain the green vector.

In the example, the chosen coefficients allowed for a simple geometric construction. However, for a general convex combination, the following approach should be used. Consider two normalised vectors  $\mathbf{a}$  and  $\mathbf{b}$  with an angle  $\varphi \neq 0$  between them. The objective is to compute the vector  $\mathbf{c}$  at an angle  $t\varphi$  with respect to  $\mathbf{a}$ , with  $t \in [0, 1]$ , expressed as a weighted sum of  $\mathbf{a}$  and  $\mathbf{b}$ ; see Figure 5.9.

Assuming without loss of generality that  $\mathbf{a}$  coincides with the horizontal axis, consider an auxiliary vector  $\mathbf{v}$  that coincides with the vertical axis so that  $\mathbf{c} = \cos(t\varphi)\mathbf{a} +$



**Figure 5.9:** *Slerping illustrated: the vector  $c$  (green) is a spherical linear combination of the vectors  $a$  and  $b$  (black).*

$\sin(t\varphi)v$ . Next, express  $b$  in a similar way as  $b = \cos(\varphi)a + \sin(\varphi)v$ . This yields

$$v = \frac{b - \cos(\varphi)a}{\sin(\varphi)},$$

which then gives

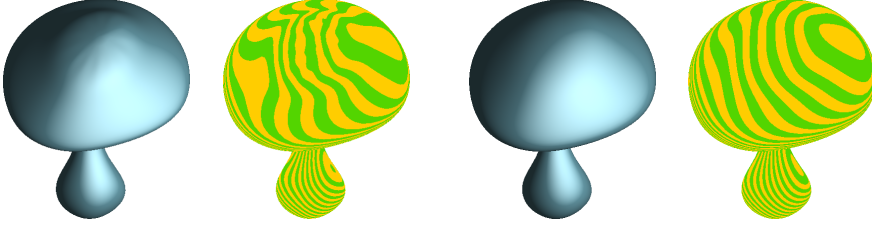
$$\begin{aligned} c &= \cos(t\varphi)a + \frac{\sin(t\varphi)(b - \cos(\varphi)a)}{\sin(\varphi)} \\ &= \frac{(\sin(\varphi)\cos(t\varphi) - \sin(t\varphi)\cos(\varphi))a + \sin(t\varphi)b}{\sin(\varphi)} \\ &= \frac{\sin(\varphi - t\varphi)a + \sin(t\varphi)b}{\sin(\varphi)} \\ &= \frac{\sin((1-t)\varphi)a + \sin(t\varphi)b}{\sin(\varphi)}. \end{aligned}$$

In the case of a convex combination of three *non-planar* unit vectors (e.g. points on the unit *sphere*), spherical barycentric coordinates [LBS06] can be used (though note that these cannot satisfy all conditions met by ordinary barycentric coordinates [Flo15]). The general approach of any number of points on the unit sphere leads to spherical averages [BF01], which requires an iterative approach.

It is important to note that for models which are eventually manufactured (i.e. made into physical objects), such as car-, boat- or airplane parts, only the actual normal fields should be used. However, for models that remain digital, different normal fields can be used. This can be applied to add details to a surface (e.g. using bump mapping) or to locally smooth a surface.

Subdivision surfaces fall into the latter category – they are notorious for the artefacts that occur around EVs. When using such objects in an animated movie or computer game, instead of using the actual normal field  $n_G$  of the geometry (i.e. the limit surface), a *subdivided* normal field  $n_S$  can be used. The idea of *subdivision shading* as

proposed in [AB08] is to take the normals of a (possibly subdivided) control net and use the subdivision stencils to subdivide these normals (using the concept of spherical averages). It follows that  $\mathbf{n}_S$  is  $C^1$  continuous. An example comparing the use of the two normal fields is shown in Figure 5.10.



**Figure 5.10:** Catmull–Clark subdivision applied to a mushroom mesh from BLENDER’s open movie *Big Buck Bunny*. The high valency  $n = 12$  at the top results in so-called polar artefacts, which are clearly visible in both the rendering and the isophotes when using the actual normal field  $\mathbf{n}_G$  (left). Using the subdivided normals  $\mathbf{n}_S$  visually removes the artefacts (right).

For many meshes, however, simply using  $\mathbf{n}_S$  everywhere yields results that are *too* smooth — detail is lost. Although this can to some extent be remedied by further subdividing the control net before subdividing the normals, a more versatile method was proposed in the original work [AB08] based on blending  $\mathbf{n}_G$  and  $\mathbf{n}_S$ . The approach is to assign scalar weights to the control points in the control net (1.0 to EVs and 0.0 elsewhere) and either bilinearly interpolate these weights over the mesh, or subdivide them using the subdivision stencils. Unfortunately, the former approach results in sharp transitions in the blended normal field, whereas the latter results in weights that are in general globally below 1.0 for approximating subdivision schemes such as Catmull–Clark and Loop. In other words, both approaches yield a blended normal field that is merely  $C^0$ . Our aim is to construct a blended normal field  $\mathbf{n}_B$  that is  $C^1$ ,

$$\mathbf{n}_B = (1 - b^p)\mathbf{n}_G + b^p\mathbf{n}_S, \quad (5.4)$$

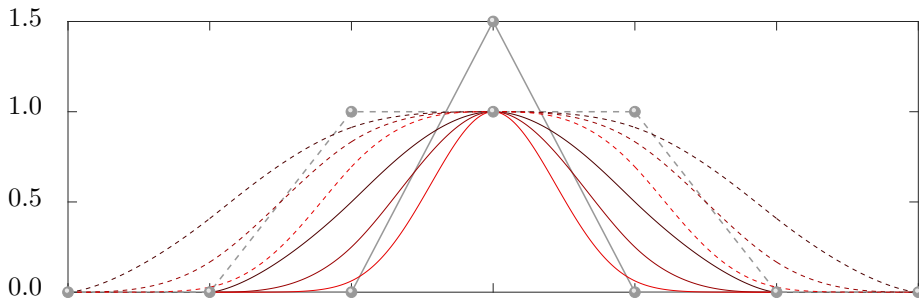
with  $b$  a suitable blending function (to be discussed) and  $p$  a positive scalar parameter. The purpose of  $p$  is to tweak the rate of decay of  $b$  around EVs, and can be set either locally (i.e. different values of  $p$  at different EVs) or globally. Note that  $p$  does not influence the support of  $b$ .

As  $\mathbf{n}_G$  is  $C^0$  at the limit positions of EVs, we need to make sure that  $b^p = 1.0$  at these points. Since  $p$  is required to be positive, it follows that  $b = 1.0$  at the limit positions. Furthermore, as  $\nabla \mathbf{n}_G$  might not be well-defined at the limit positions of EVs, all terms in the gradient of (5.4) containing  $\nabla \mathbf{n}_G$  should vanish at these points. For the full proof, we refer to [BBK18a]. Based on this, we propose the following two methods:

- A.) In case of separated EVs (i.e. at most one EV per face in the mesh), we can assign weights  $w_n$  to the EVs and zero weights elsewhere such that applying (generalised) limit stencils results in values of 1.0 at the limit positions of EVs. This results in nonzero values of  $b$  in the two-ring neighbourhood of EVs.

B.) For EVs (or EFs) that are *not* separated, approach A.) might not be applicable. An alternative is to assign weights 1.0 to the EVs as well as all vertices in their one-ring neighbourhoods, which again results in values of 1.0 at the limit positions of EVs. This results in nonzero values of  $b$  in the three-ring neighbourhood of EVs.

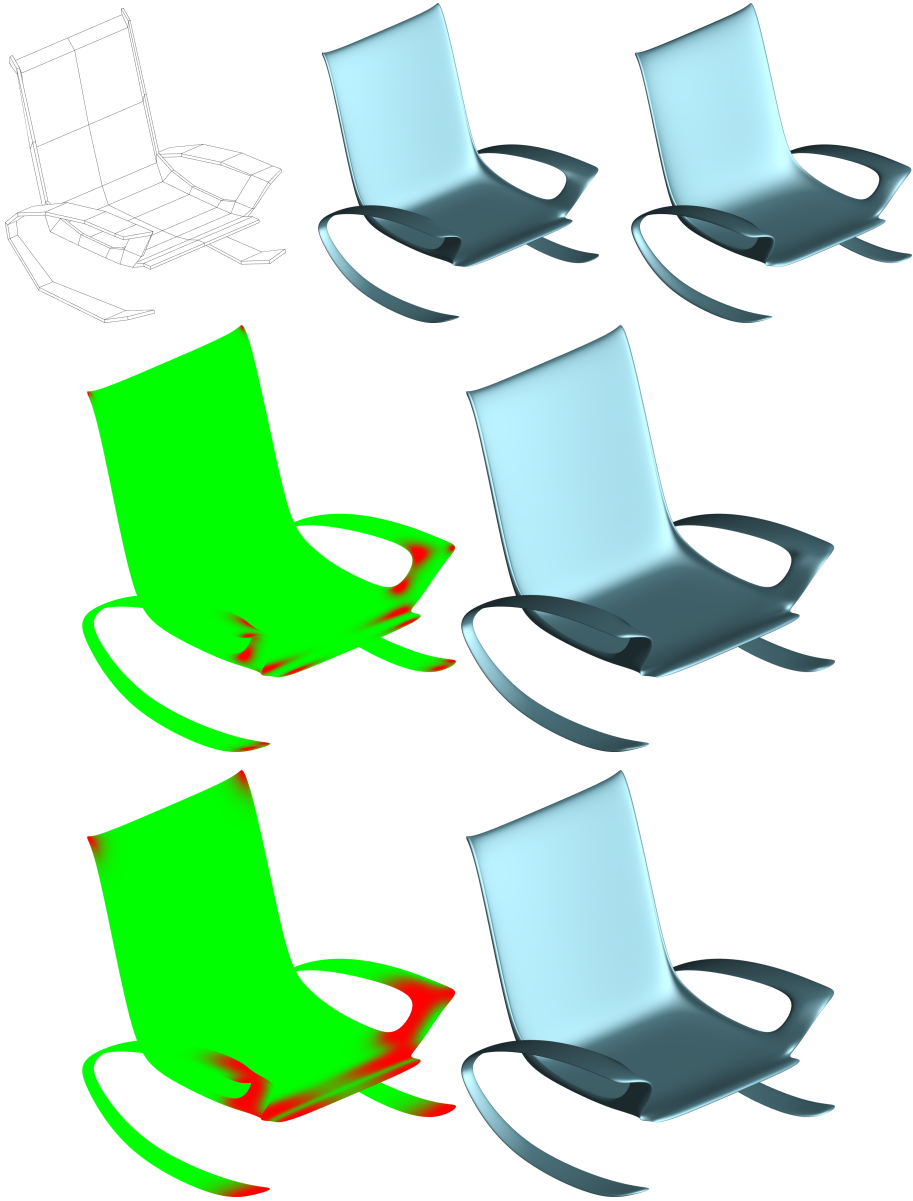
Figure 5.11 illustrates the support of  $b$  for the two approaches for various values of  $p$  in a univariate setting.



**Figure 5.11:** The univariate equivalents of the two types of blending approaches, with A.) solid and B.) dashed, raised to different powers  $p \in \{0.5, 1.0, 2.0\}$  shaded from dark red to bright red. The control nets are shown in grey.

Finally, Figure 5.12 compares the two approaches. Although approach A.) yields satisfactory results for some objects, based on these and other results, B.) is the overall preferred approach and comes with the added benefit that it is straightforward to implement.





**Figure 5.12:** Chair model rendered as a Catmull–Clark surface using  $\mathbf{n}_G$  and  $\mathbf{n}_S$  (top). The former results in lighting artefacts near the front, whereas the latter removes detail such as the ridge on the right. Approach A.) does not manage to remedy the artefacts (middle), whereas approach B.) does (bottom). The associated blending functions  $b$  are shown colour-coded with red indicating  $b = 1$  and green  $b = 0$ . A value of  $p = 1$  was used throughout.

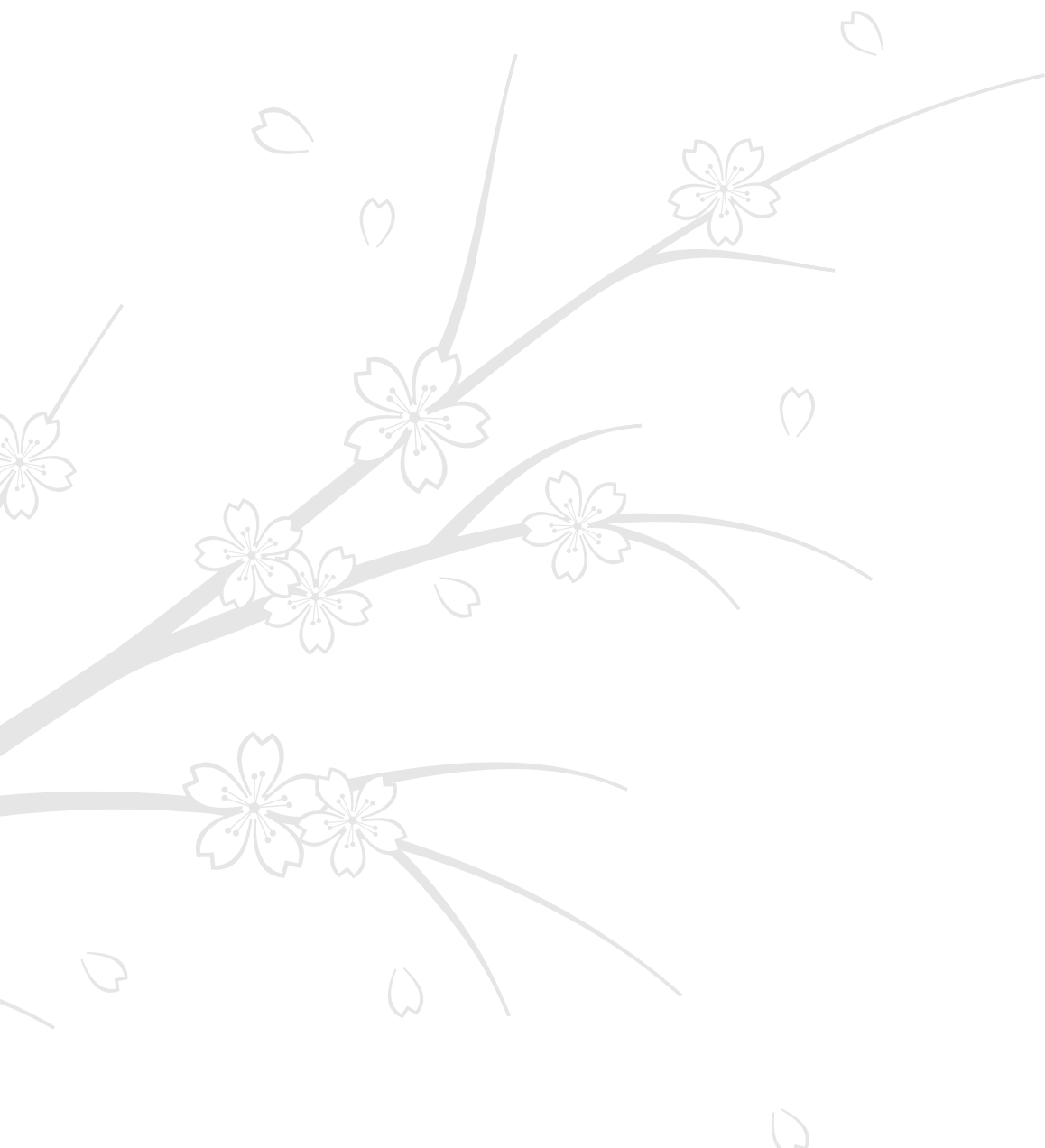


## 6 | Colour gradients in vector graphics



Parts of this chapter have been published as

- Pieter J Barendrecht, Martijn Luinstra, Jonathan Hogervorst and Jiří Kosinka. “Locally refinable gradient meshes supporting branching and sharp colour transitions”. In: *The Visual Computer* 34.6-8 (2018), pp. 949–960. See Section [6.2](#).



In this last core chapter we discuss two techniques that can be used to create (almost) photorealistic illustrations in the context of vector graphics. *Gradient meshes* combine curve networks with colour values, which in the most general setting leads to colour gradients on smooth surfaces of arbitrary manifold topology supporting local refinement. *Diffusion curves* focus on the diffusion of colour gradients assigned to spline curves, which can be solved for using FEM or BEM, or alternatively, approximated with ray-tracing.

## 6.1 Primitives

Vector graphics is based on the use of building blocks referred to as *primitives*, whose properties remain editable after their initial creation. This allows for a flexible workflow that is fundamentally different compared to that of raster graphics, which is pixel-based. The list of primitives includes basic shapes such as rectangles and ellipses, Bézier curves and text, all of which can be described as *paths*. Rather than representing these paths in terms of pixels, they are defined as mathematical objects, which makes them resolution invariant (i.e. scalable without losing quality). Boolean operations and offsets of paths help the designer to model an envisioned shape.

Although paths are usually expressed in the form of cubic Béziers, alternatives do exist, including B-spline- and spiro curves. The search for a spline with even better characteristics (e.g. more intuitive from the user's point of view) is an area of active research — a recent example is the introduction of the  $\kappa$ -curve [Yan+17].

Ultimately, two of the most important attributes of a design are its shape and its colour. It is clear that the shape can be described using paths; colour is another matter. Fills can be flat (i.e. solid) or based on various types of gradients such as linear and radial gradients. However, experience shows that more intricate designs often require the use of something more sophisticated. As such, following the introduction of *gradient fills* in PostScript 3 [Ado97; Ado99], the gradient mesh was introduced [Ado98], later followed by diffusion curves [Orz+08]. Gradient meshes are discussed in detail, along with multiple improvements; diffusion curves are considered from the perspective of existing literature.

## 6.2 Gradient meshes

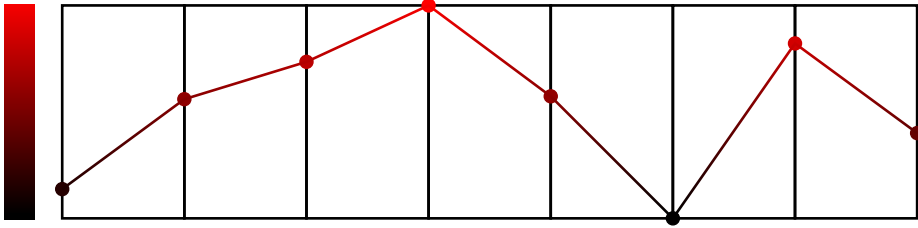
The starting point of the traditional gradient mesh primitive is a planar cubic curve network (see Section 3.1.1) with the topology of a Cartesian grid. Each cell in the grid is therefore defined by four cubic Béziers. Although *both* the control points and tangent handles are editable, in the following we assume the composite curves in the network to be  $C^1$  continuous.

Next, let us consider the colour aspect of the primitive. We only allow the user to assign colours to control points, *not* to tangent handles. Assuming an sRGB colour space for now, each colour channel<sup>†</sup> can be considered separately as a *height map* living on the 2D network. Focusing solely on the red channel, this raises the question how the red values should be interpolated between the control points. A straightforward approach is to use linear interpolation, as shown in Figure 6.1. Note that it depicts the colour interpolation in the *functional* setting (also referred to as the *non-parametric* setting) — that is, the piecewise linear Bézier  $r_i(t)$  representing the red values (vertical axis) is plotted against the parameter  $t$  (horizontal axis).

Although this might be a reasonable approach in some cases, it results in a colour propagation that is merely  $C^0$  across the network curves. Smooth interpolation is often desired, though in the context of colour it is not directly clear what is meant by *smooth*. In this work we aim at an interpolation that is at least  $G^1$ , which we conjecture is a

---

<sup>†</sup>In addition, an *alpha* channel (transparency/opacity) could be considered.



**Figure 6.1:** Piecewise linear interpolation of the red values assigned to control points of a piecewise cubic Bézier.

sufficient condition.

With this in mind, we move on to quadratic and higher-order interpolation of colour. In these cases, tangent handles come into play, which means that we need to determine to which positions in parameter space these handles correspond. Let us consider a quadratic Bézier  $r_q(t) = \sum_{k=0}^2 r_k B_k^2(t)$ , with  $r_k$  the red values. Note that  $r_1$  is a degree of freedom (DoF) as no value has been assigned to it yet. Additionally, in this functional setting we have  $t = \sum_{k=0}^2 t_k B_k^2(t)$ . Using a parameterisation on the unit domain, by the end-point interpolation property of Béziers it is clear that  $t_0 = 0$  and  $t_2 = 1$ . The value of  $t_1$  can be obtained by expressing the Bernsteins in the power basis, storing their coefficients as columns of a matrix, and solving the resulting system

$$\begin{pmatrix} 1 & t & t^2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} 1 & t & t^2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad (6.1)$$

which shows that  $t_1 = \frac{1}{2}$ . In other words, in the functional setting, the tangent handles of a piecewise quadratic Bézier curve correspond to the midpoints of the parameter intervals. Figure 6.2 shows two such curves with their  $x$ - and  $y$ -components visualised individually as functional curves.

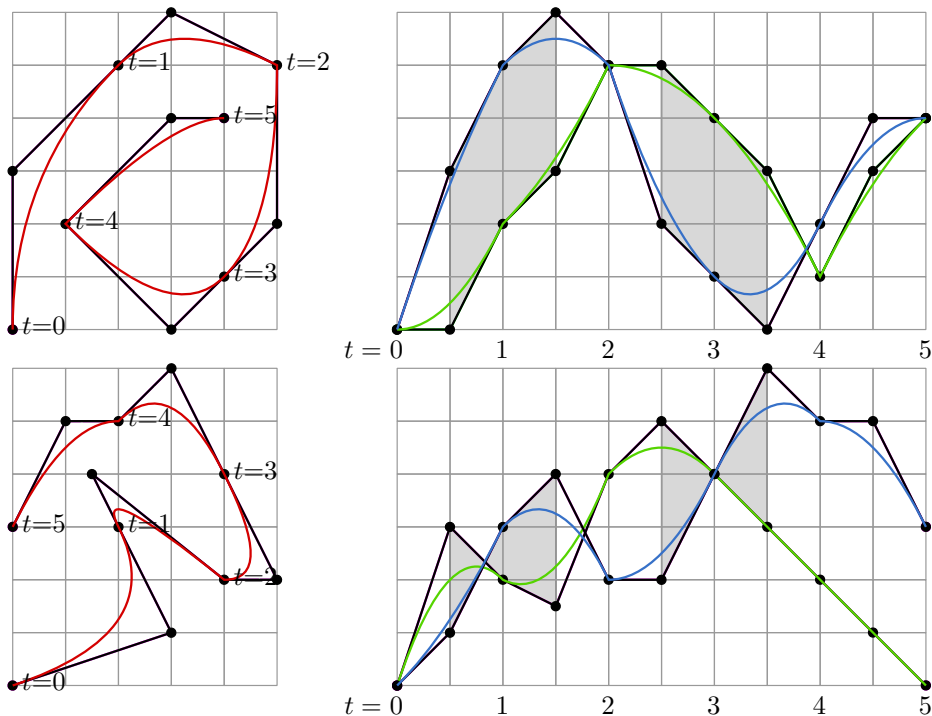
Given the red values at control points, we can now attempt to assign red values to the handles such that the quadratic Béziers connect with  $G^1$  continuity. Although this is certainly possible in some cases, for others (e.g. strongly oscillating values or adjacent values on opposite borders of the colour gamut) it is not. Furthermore, observe that changing the angle of a tangent handle in one segment propagates throughout the entire curve, i.e. has a global effect.

Consequently, we move on to piecewise cubics. The parameter values associated with the tangent handles are readily determined using the same approach,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad (6.2)$$

which yields  $t_1 = \frac{1}{3}$  and  $t_2 = \frac{2}{3}$ . It can be shown that handles (or more general, interior control points) of a Bézier of general degree  $d$  map to parameter values  $\frac{k}{d}$  [Far02]<sup>†</sup>.

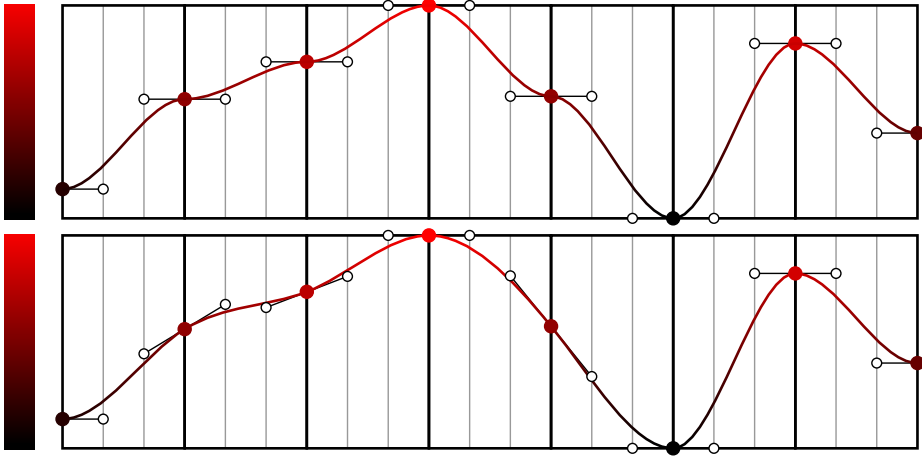
<sup>†</sup>Note that this also facilitates the construction of control polygons of the Bernsteins themselves.



**Figure 6.2:** Two piecewise quadratic Bézier curves (left) and their  $x$ - and  $y$ -components visualised individually as functional curves in green and blue, respectively (right). Both curves are  $G^1$  at  $t = 1$  and  $C^1$  at  $t = 3$  (indicated using grey highlights on the right). Additionally, one of the components is  $C^1$  at  $t = 4$ .

For functional  $C^1$  continuity between Bézier segments, the tangent handles on both sides of a control point have to be co-linear with it (note that in the functional setting, there is no such thing as  $G^1$  continuity). This provides us with a DoF per control point, namely the *angle* of these handles. Clearly, a monotonic interpolation is desired. The easiest approach simply sets all angles to  $0^\circ$ , which prevents the resulting (red) values from escaping their range  $[0, 1]$ . Alternatives are often based on minimizing e.g. the bending energy of the curve [WA02; LS09], which yields visually smoother results. Such an approach can be used for sequences of red values that are monotonous — at control points which separate the regions of monotonicity, the angle should nevertheless be  $0^\circ$  to ensure colour values that are within the gamut. Figure 6.3 illustrates both approaches.

This completes the assignment of colour values to the cubic curve network. Finally, each cell can be interpreted as a bicubic patch, resulting in the interpolation of colour in the interior. Without information about the interior points, a Ferguson patch is the most straightforward approach. However, it is well-known that for a *geometric* interpolation of a cubic curve network, the zero twist-vectors associated with Ferguson patches lead to flat spots for non-translational surfaces [Far02]. As such, a short study



**Figure 6.3:** Piecewise cubic interpolation of the red values assigned to control points. In this example, there are four regions of monotonicity. The angle of the tangent handles at control points separating these is set to  $0^\circ$ . One option is to do the same for all remaining control points (top), whereas another choice is to optimise these angles by e.g. minimizing the bending energy (bottom).

to the assignment of general twist-vectors is worthwhile.

Consider a control point  $\mathcal{O}$  in the curve network with an ordinary valency  $n = 4$  and two pairs of handles – see Figure 6.4 – such that  $A - \mathcal{O} = \mathcal{O} - C$  and  $B - \mathcal{O} = \mathcal{O} - D$ . In other words, the tangent handles are pairwise co-linear with  $\mathcal{O}$ , and form a parallelogram with  $\mathcal{O}$  as its centroid. From a Bézier point of view, we can now introduce a twist vector  $E$  anywhere (not necessarily in the plane of the parallelogram), and by the condition of  $C^1$  continuity between patches, determine the other twists  $F$ ,  $G$  and  $H$ . If we now assume  $\mathcal{O}$  to coincide with the origin, we see that

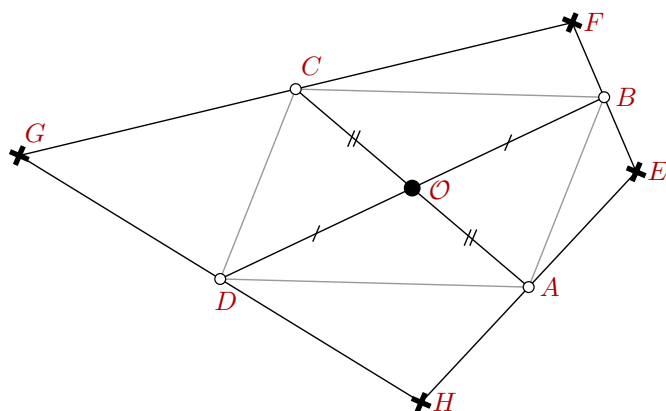
$$\begin{aligned} & E + 2(B - E) + 2(C - F) + 2(D - G) + 2(A - H) \\ &= E - 2(E + F + G + H) \\ &= E, \end{aligned}$$

which holds because  $\mathcal{O}$  is also the centroid of our (potentially skew/non-planar) quadrilateral  $EFGH$  as it lies at the intersection of its bimedians<sup>†</sup>. This shows that we have indeed 2 scalar DoFs for the assignment of the twist-vectors. Instead of vanishing twists, a good option would be to use Adini's twists; see [Far02] for details and alternatives.

Notwithstanding, our current implementation of the gradient mesh is based on  $0^\circ$  tangent handles and vanishing twists – Figure 6.5 shows an example of the interior colour propagation in a single cell. Visual comparison in GIMP shows that CORELDRAW appears to be using the same approach. In contrast, it seems that in ADOBE ILLUSTRATOR, the angle of the tangent handles is optimised, though with regard to which measure is currently unknown. INKSCAPE offers two interpolation methods, one which

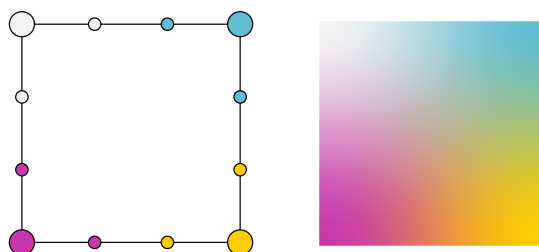
<sup>†</sup> Additionally, note that  $ABCD$  is the Varignon parallelogram of the quadrilateral  $EFGH$ .





**Figure 6.4:** A control point  $O$  with its two pairs of tangent handles, forming the parallelogram  $ABCD$ . The twists  $EFGH$  define a quadrilateral.

mimics ILLUSTRATOR's approach, and another one which uses bilinear interpolation. Figure 6.6 compares the approaches.

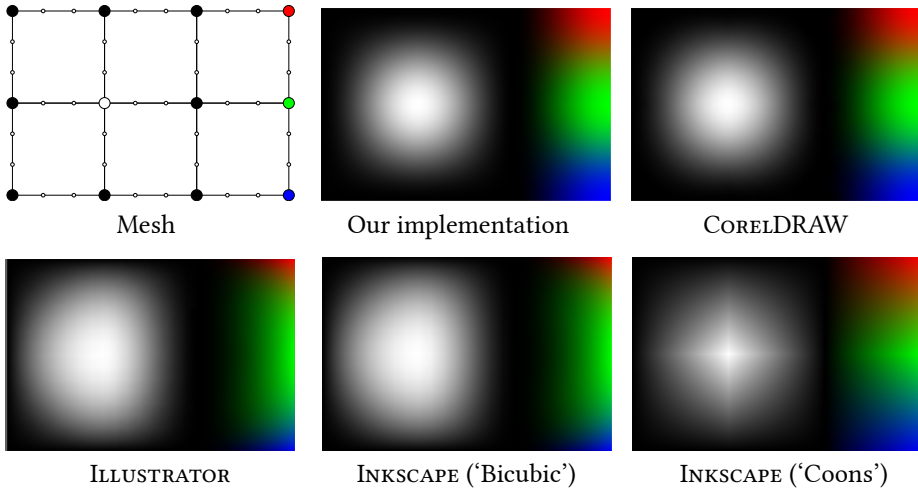


**Figure 6.5:** Colour information on the boundary (left). The control points (large circles) each have a different colour assigned to them. Because of the  $0^\circ$  tangent handles (small circles), the handles copy the colour of their associated control point. Using zero twists (in Hermite sense), an interior colour propagation is realised (right).

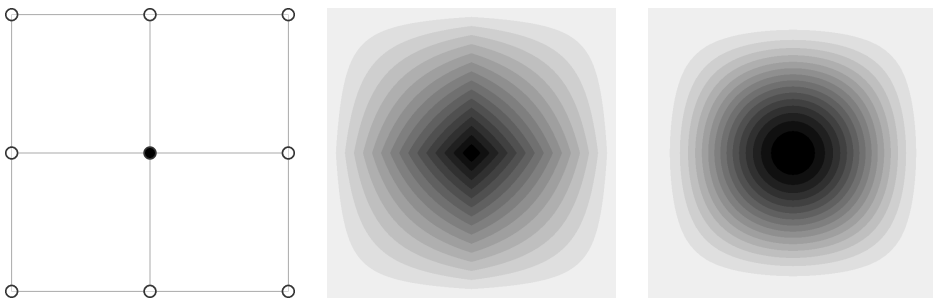
## 6.2

Summarising, the geometry of the cubic curve network can be fully determined by the designer. Colour can only be assigned to the control points; at tangent handles it is computed. Finally, the interior colour propagation requires the computation of twist-vectors for both geometry and colour.

Although we assumed  $C^1$  continuity between Béziers, in practice there is the occasional need for sharp transitions. This is fine, but clearly results in  $C^0$  continuity of colour. As this reduced continuity can be more difficult to spot compared to a purely geometric setting, we use colour banding to visualise iso-curves in colour space. Figure 6.7 illustrates the approach for a bilinear interpolation of colour (cf. Figure 6.1) versus our bicubic interpolation as described above.



**Figure 6.6:** Comparing renderings of a  $2 \times 3$  gradient mesh in different software packages. Handles are at their default location and are visualised as small open circles.



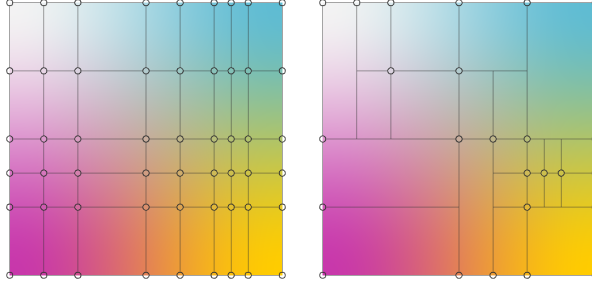
**Figure 6.7:** An elementary gradient mesh consisting of  $2 \times 2$  cells. All control points are assigned a white colour with the exception of the interior control point, which is black (left). Bilinear interpolation of the colour naturally results in  $C^0$  continuity (middle), whereas our bicubic interpolation yields  $C^1$  continuity (right). Colour banding has been used to highlight the iso-curves in colour space.

### 6.2.1 Local refinement

Upon using the gradient mesh primitive for a while, a couple of shortcomings present themselves. One of these is related to the refinement of the curve network — local detail requires the insertion of additional control points, which in the traditional case results in the insertion of entire rows and/or columns of control points. This quickly clutters the canvas with undesired points, and slows down the workflow considerably.

Clearly, the gradient mesh would benefit from a *local refinement* approach. Our first attempt towards this improvement was to locally quadrisect patches, but we soon realised that true local refinement should be based on *bisecting* individual patches instead. We now take a look at this in the dyadic sense, that is, splitting patches at  $u = \frac{1}{2}$  or  $v = \frac{1}{2}$  (see Figure 6.8), and ultimately extend this to bisecting patches at arbitrary

parameter values.



**Figure 6.8:** Global refinement (left), where each refinement results in an entire row and/or column of control points, is the only form of refinement available in existing implementations of the gradient mesh primitive. We propose a local refinement approach (right). Note that the meshes define identical colour surfaces.

From a mathematical point of view, bisecting patches comes down to the subdivision of two opposite curves of the patch and the computation of interior data. This results in new control points and/or *T-sections* – each with their associated tangent handles – and in new twist vectors, which are generally *nonzero* (and hence we require a full bicubic patch description instead of Ferguson patches). Clearly, de Casteljau’s algorithm could be used to compute the new points and handles on the boundary curves, though as we are also dealing with twists, a Hermite description can be more convenient. Recall that a cubic curve can be expressed in Bézier or Hermite form as

$$C(t) = \sum_{k=0}^3 \mathbf{P}_k B_k(t) = \sum_{k=0}^3 \mathbf{Q}_k H_k(t), \quad (6.3)$$

where  $\mathbf{Q}$  can be written in terms of  $\mathbf{P}$  as

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{P} = \mathbf{X} \mathbf{P}.$$

This also implies  $\mathbf{P} = \mathbf{X}^{-1} \mathbf{Q}$ , so that

$$\mathbf{P}^T \mathbf{B}(t) = (\mathbf{X}^{-1} \mathbf{Q})^T \mathbf{B}(t) = \mathbf{Q}^T \boxed{\mathbf{X}^{-T} \mathbf{B}(t)} = \mathbf{Q}^T \boxed{\mathbf{H}(t)},$$

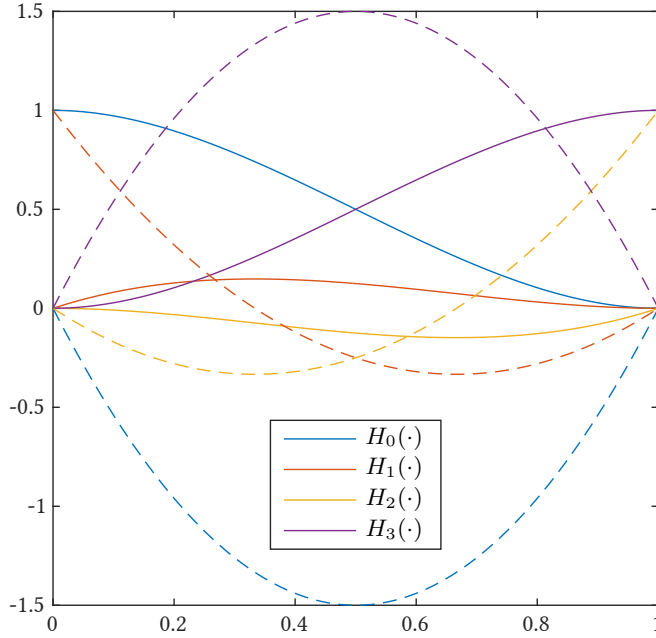
which in turn shows that  $\mathbf{B}(t) = \mathbf{X}^T \mathbf{H}(t)$ . Evaluating position, tangents (i.e. first order partial derivatives) and twists (i.e. mixed partials) in Hermite form now follows as

$$\begin{aligned} S(u, v) &= \mathbf{H}^T(u) \mathbf{C}_H \mathbf{H}(v), & \frac{\partial S(u, v)}{\partial u} &= \mathbf{H}'^T(u) \mathbf{C}_H \mathbf{H}(v), \\ \frac{\partial S(u, v)}{\partial v} &= \mathbf{H}^T(u) \mathbf{C}_H \mathbf{H}'(v), & \frac{\partial^2 S(u, v)}{\partial u \partial v} &= \mathbf{H}'^T(u) \mathbf{C}_H \mathbf{H}'(v), \end{aligned} \quad (6.4)$$

with

$$\begin{aligned} \mathbf{C}_H &= \begin{pmatrix} S(0,0) & S_v(0,0) & S_v(0,1) & S(0,1) \\ S_u(0,0) & S_{uv}(0,0) & S_{uv}(0,1) & S_u(0,1) \\ S_u(1,0) & S_{uv}(1,0) & S_{uv}(1,1) & S_u(1,1) \\ S(1,0) & S_v(1,0) & S_v(1,1) & S(1,1) \end{pmatrix} \\ &= \mathbf{X} \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{pmatrix} \mathbf{X}^T = \mathbf{X} \mathbf{C}_B \mathbf{X}^T. \end{aligned}$$

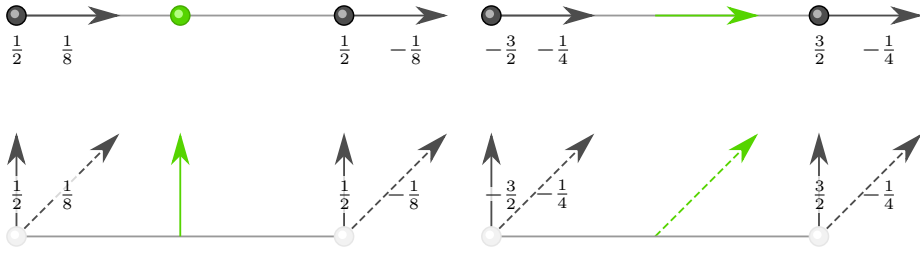
In our dyadic setting, evaluation of  $\mathbf{H}(\cdot)$  and  $\mathbf{H}'(\cdot)$  – see Figure 6.9 – results in a set of four stencils per boundary curve, which are shown in Figure 6.10.



**Figure 6.9:** The cubic Hermite basis functions and their derivatives (shown using the same colours but dashed).

Bisecting a patch means to parameterise each half on  $[0, \frac{1}{2}] \times [0, 1]$ . Let us focus on a single cubic curve  $C(t)$  that is split into two. With  $t \in [0, \frac{1}{2}]$ , we can then define Hermite-like basis functions that describe the first half as

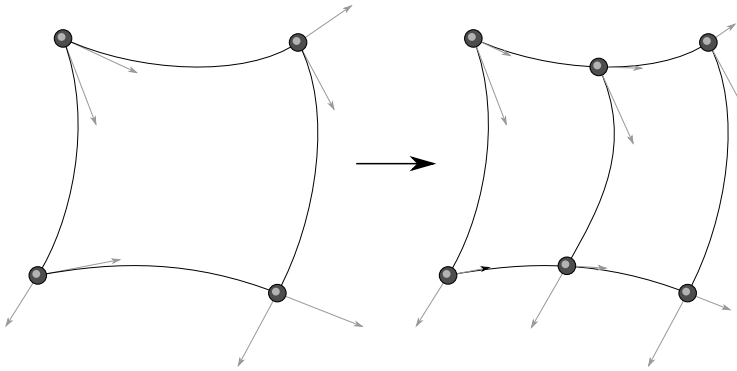
$$\begin{aligned} C(t) &= (1 - 12t^2 + 16t^3)C(0) + (t - 4t^2 + 4t^3)C'(0) + \\ &\quad (-2t^2 + 4t^3)C'\left(\frac{1}{2}\right) + (12t^2 - 16t^3)C\left(\frac{1}{2}\right) \\ &= H_0(2t)C(0) + \frac{1}{2}H_1(2t)C'(0) + \frac{1}{2}H_2(2t)C'\left(\frac{1}{2}\right) + H_3(2t)C\left(\frac{1}{2}\right). \end{aligned}$$



**Figure 6.10:** The stencils for the computation of position, tangents (solid arrows) and twist (dashed arrow) at  $u = \frac{1}{2}$  on the curve  $v = 0$ . Stencils for the other curves are virtually identical.

In other words, if we scale the tangents  $C'(0)$  and  $C'(\frac{1}{2})$  by a factor  $\frac{1}{2}$ , we can render the curve using the familiar cubic Hermite basis functions  $H_k(s)$  with  $s = 2t$ ; note that the correct derivative is the one with respect to  $t$ , i.e.  $\frac{dH_k(s)}{dt} = 2H'_k(s)$ . Naturally, the same holds for the second half of  $C(t)$ . In Bézier form, we obtain an equivalent result – the de Casteljau algorithm shows that the original tangent handles get scaled by a factor  $\frac{1}{2}$ .

In practice, this means that we have to scale the versal tangent and twist-vector computed at  $u = \frac{1}{2}$  (using the stencils from Figure 6.10) – as well as the existing versal tangents and twist-vectors at  $u = 0$  and  $u = 1$  – by a factor  $\frac{1}{2}$ . The positions and transversal tangents do not require scaling. See Figure 6.11 for an example.



**Figure 6.11:** Splitting a patch using the stencils from Figure 6.10. The twist vectors are not shown.

Locally splitting patches eventually results in T-sections<sup>†</sup>. The position or colour of these T-sections *cannot* be updated by the user, as this would result in  $C^{-1}$  discontinuities. Instead, T-sections inherit their information from the curve that they live on, and are updated when this curve is modified. Consider e.g. the T-sections  $T_1$  and  $T_2$  that live on the curve connecting the control points  $P_1$  and  $P_2$  in Figure 6.12. The same

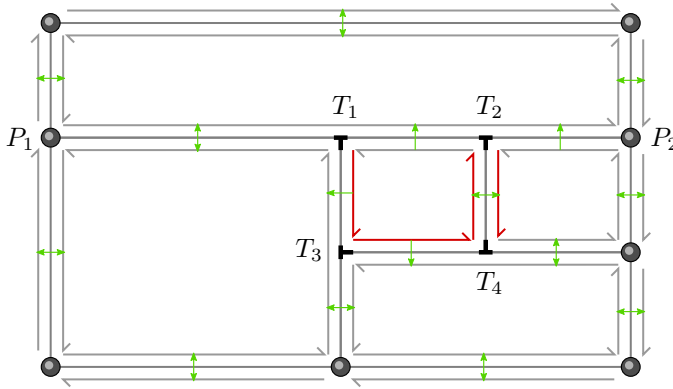
<sup>†</sup>Which are known as *hanging nodes* in an FEM context.

holds for the tangent handles of a T-section that are tangent to this curve. However, the position of the tangent handle transversal to this curve *can* be changed by the designer.

T-sections become editable if they lie on the boundary of the mesh, or if the adjacent patch is split at the parameter value the T-section corresponds to — in both cases, they reduce to ordinary control points.

The appearance of T-sections also indicates that we require an appropriate data structure. After all, the topology of a locally refined mesh no longer coincides with that of a Cartesian grid. One option would be to use a list of binary trees, also referred to as a binary forest. The initial patches would be the roots in this case — every bisection of a patch would result in two new branches (which would be the leaves of that tree at that point).

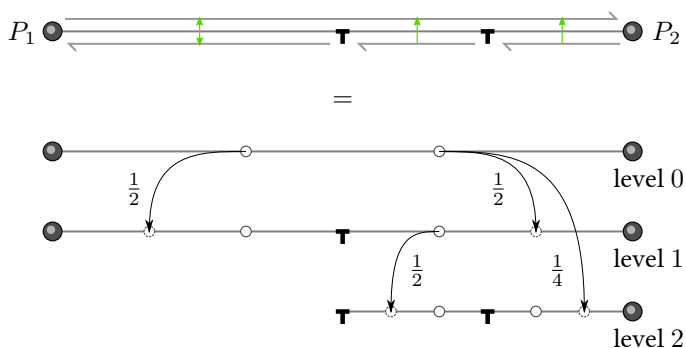
Instead, we choose to use an augmented half-edge data structure (see Appendix B), which comes with built-in support for meshes of arbitrary manifold topology. Figure 6.12 shows a mesh containing several T-sections and highlights several differences with respect to the traditional half-edge structure.



**Figure 6.12:** A mesh with T-sections. Half-edges are indicated in grey. Green arrows indicate the opposite of a half-edge, which is not always a symmetrical relation near T-sections. T-sections have fixed links to the outgoing half-edge pointing in the direction of their stem, which are here indicated in red.

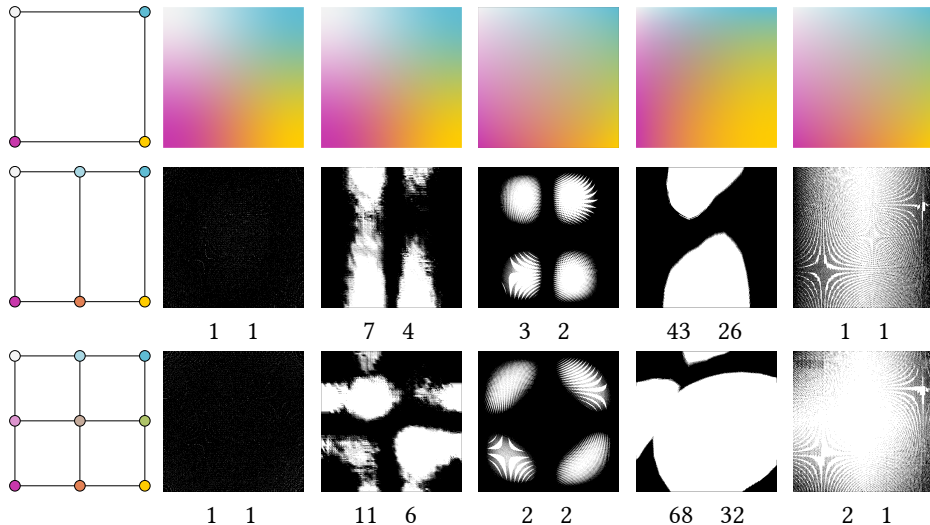
In our implementation, we choose to share tangent handles between refinement levels. Figure 6.13 shows how a single curve is bisected twice — the three resulting curves all re-use existing tangent handles.

The refinement approach as discussed above is *mathematically exact*. That is, upon refinement, neither the geometry nor the colour changes. We perceive this as the obvious approach, though when comparing our refinement to the existing implementations in software packages, it turns out that none of the latter are mathematically exact, which means that after refinement the colour propagation is (subtly) different. Figure 6.14 compares the results of bisection and quadrissection. The differences between the original rendering and bisection or quadrissection results have been thresholded in GIMP to the point where white pixels appear. Slightly lower threshold values often help to highlight the deviations. From the resulting thresholded differences, CORELDRAW



**Figure 6.13:** Sharing handles between pairs of half-edges of different levels. The original handles are re-used and scaled by a factor  $\frac{1}{2}$  after the first refinement, also resulting in two new handles on the sides of the T-section. One of these is re-used and scaled after the second refinement, whereas the other reused handle is an original one scaled by a factor  $\frac{1}{4}$ .

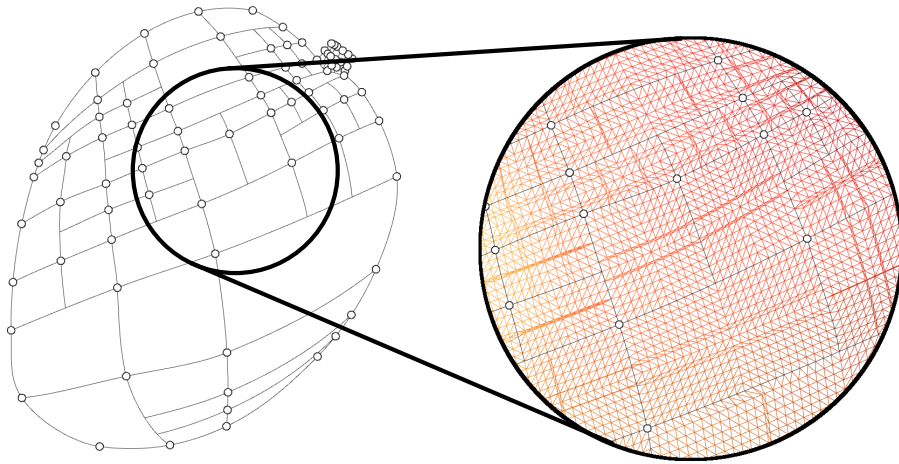
appears to have issues regarding the computation of the new tangent handles. In the case of ILLUSTRATOR, we surmise its twist-vectors are slightly off. The ‘bicubic’ option in INKSCAPE has several serious issues; in contrast, the ‘Coons’ option is almost exact, but shows rendering artefacts. Note that also in our case we might have minor rendering artefacts in cases where the tessellation is not pixel-accurate.



**Figure 6.14:** Comparing refinement accuracy. From left to right: our implementation, CORELDRAW, ADOBE ILLUSTRATOR, INKSCAPE (‘Bicubic’) and INKSCAPE (‘Coons’). The top row shows the rendering of an elementary gradient mesh (consisting of a single patch), the middle and bottom rows the thresholded differences after refining vertically and in both directions, respectively. The two values below each result are thresholding values (minimal value and selected value for better visual comparison).

To conclude our discussion regarding the dyadic splitting of individual patches, some remarks regarding the tessellation are in order (see also Section 5.2). First, we use an adaptive tessellation approach which aims at creating a triangulation with triangle edges the length of a user-controllable number of pixels. Using fractional even tessellation levels minimises swimming artefacts when zooming (i.e. it facilitates smooth transitions). Secondly, in order to guarantee crack-free tessellation, the triangle edges on one side of a curve should match those on the opposite side of the curve. In the case of local refinement, we compute an appropriate even integer tessellation level for the curve of the highest *curve* level (see e.g. Figure 6.13); tessellation levels for the curves of lower curve level are then multiplied by the appropriate powers of 2. Figure 6.15 shows an example.

Most OpenGL drivers set the maximum tessellation level to 64. This leaves us with a choice of either virtually subdividing patches when higher levels are required, or accepting larger triangles yielding renderings that are not pixel accurate.

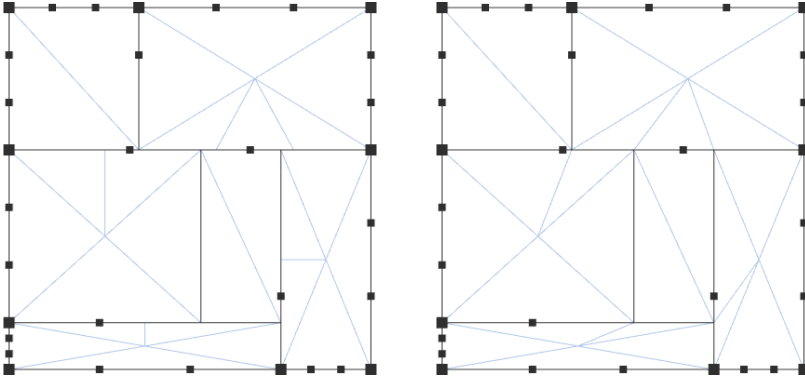


**Figure 6.15:** Locally refined gradient mesh of a mango (left) and its tessellation at a certain number of pixels per triangle edge (right). Note the smooth transition between different sides of patches facilitated by the fractional tessellation levels.

In the setting where individual patches can be split at *arbitrary* parameter values, there are a couple of differences compared to the dyadic approach discussed above. First, stencils for the computation of position, tangent handles and twist-vectors are no longer static, but need to be computed for each case individually using (6.4). The computed values have to be scaled accordingly, but can still be shared between different levels of refinement in some cases. In addition, a different data structure is used — like in a traditional half-edge structure, the half-edges now only appear in pairs. The T-sections become instrumental in traversing the mesh. With regard to tessellation, the challenge here is to maintain a crack-free result. Our approach is to save the parameter values of the T-sections with regard to the curve they live on, and use this to determine the correct integer tessellation levels for the patch boundaries in the tessellation control



shader (TCS). The output of the tessellation module is uniform, which is corrected in the tessellation evaluation shader (TES). Figure 6.16 shows an example with an uncorrected and a corrected tessellation. As it concerns a preliminary implementation, the GUI is not as polished as the preceding one.



**Figure 6.16:** A mesh with T-sections at arbitrary parameter values; the large solid squares correspond to control points editable by the user, the smaller ones to the editable tangent handles. The tessellation as generated by OpenGL in `equal_spacing` mode is uniform, which results in cracks (left). In order to obtain a result that is crack-free, the positions of the points on the patch boundaries need to be corrected (right). For demonstration purposes, the tessellation levels in the example are chosen to be minimal.

Before moving on to our next improvement, we remark that local refinement is not only useful for artists in the process of manually creating new designs. It could also be used as a form of compression, possibly by merging patches in existing gradient meshes provided that they connect with the required continuity (which is  $C^\infty$  in theory, though  $C^3$  suffices in practice).

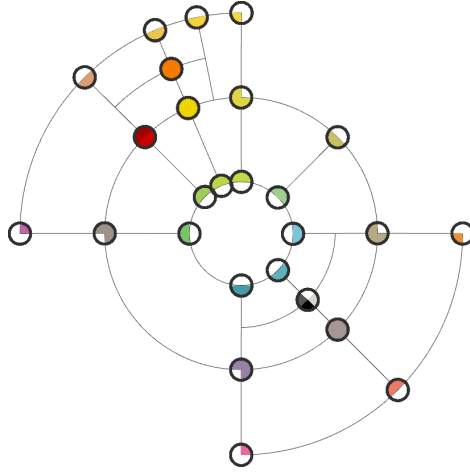
## 6.2.2 More flexible topology

• The approaches discussed in this section are still research in progress, and are expected to lead to a future publication.

Another obvious shortcoming of the traditional gradient mesh is its fixed topology, which was already briefly mentioned in the context of (local) refinement. Representing objects as deformed Cartesian grids is in many cases an artificial approach that deviates considerably from a natural workflow. A common workaround is to overlap multiple gradient meshes, but this makes editing unnecessarily complicated.

A straightforward improvement is to allow *branching*, that is, local extrusions of the gradient mesh. Note that this also enables the creation of truly cyclic meshes in the sense that these re-use the initial and final vertices instead of overlapping two copies; see Figure 6.17.

The next step is to consider meshes of arbitrary manifold topology. A natural approach seems to consider quad-dominant meshes containing the occasional triangular



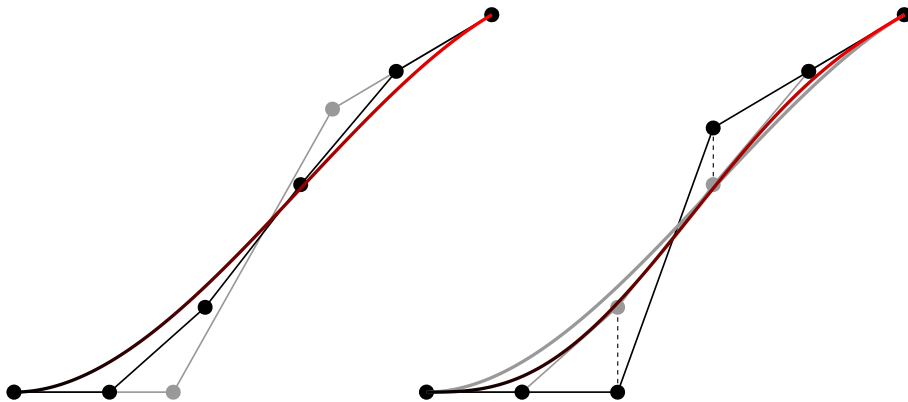
**Figure 6.17:** A cyclic gradient mesh using local refinement and branching (as well as sharp colour transitions, to be defined in the next section).

element, in which the vertices can have any valency  $n \geq 3$ . There are various options to choose from here, ranging from patch-based approaches using Bézier or Gregory descriptions to subdivision surfaces.

As we saw in Chapter 3, interpolating a given cubic curve network with Bézier patches such that the overall result is  $G^1$  continuous generally requires a combination of biquintic patches and sextic triangular ones. However, recall that for even valencies, the vertex enclosure constraint (VEC) might not be satisfied by the degree-elevated curve network; see also (3.21). Assuming the cubic curves have been degree-elevated to quintics for both geometry and colour, observe that the former is fixed — modifying the geometry is in most cases undesirable. However, in the case of colour, we *do* have some DoFs — the values associated with the two innermost control points, to be exact. Their default values follow from the degree-elevation, but can still be altered if necessary. One option is to try to tune the  $z$ -values (e.g. red values) such that the VEC is satisfied, though monotonicity of the curve should be ensured at all times. Alternatively, these  $z$ -values can be updated such that they become co-planar with the left- and right pair of control point and tangent handle, respectively<sup>†</sup> (see Figure 6.18). In case both tuning strategies fail to yield a satisfying result, an approximate  $G^1$  construction might still be an option, using e.g. a least squares approach.

For meshes with all valencies  $n \in \{3, 4, 5\}$ , with  $X$ -tangents (see Section 3.1.1) for all vertices with valency  $n = 4$ , theoretically there should not be any difficulties. In practice, however, there might still be hiccups. As we are in an interactive setting — new twist vectors and other interior points are re-computed almost continuously — we should ensure a smooth behaviour of the changing colour propagation. That is, moving a control point or tangent handle can result in sudden ‘jumps’ in the colour propagation in case of e.g. co-linear or coinciding curves, causing the system(s) to solve to become

<sup>†</sup>This illustrates another advantage of quintics over quartics — using the latter, we would only have one DoF, which would not allow this second kind of tuning.



**Figure 6.18:** Degree-elevation of a cubic curve to a quintic one (left) and subsequently moving the two innermost control points (i.e. red values) such that they become co-linear with the first- and last two, respectively. The original curve is shown in grey (right).

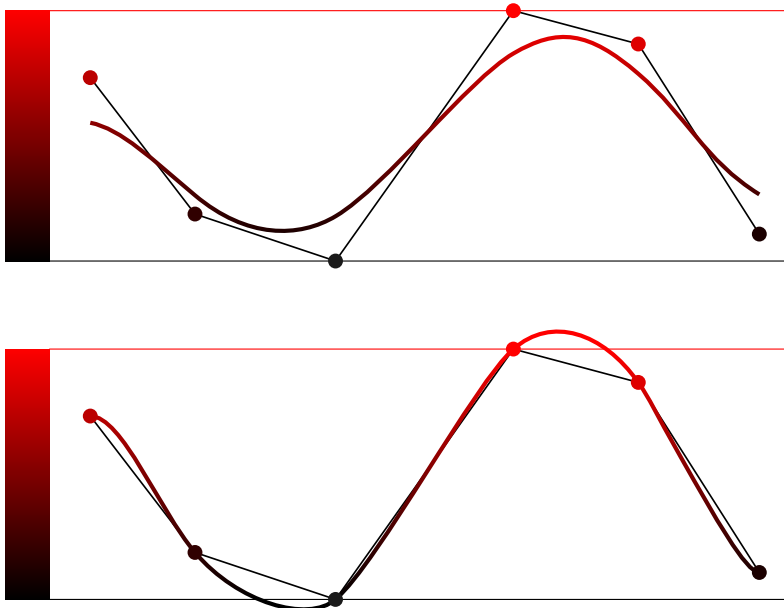
singular. Ideally, this should be avoided at all times — restricting the movement of these entities (or some other form of user-feedback) might therefore be necessary.

When discussing the interpolation of curve networks before, we mentioned a couple of alternative approaches for achieving a  $G^1$  result. Splitting (e.g. quadrisecting) patches to obtain more DoFs is a common approach, though this typically only works in the setting of interpolating *meshes* rather than curve networks. Likewise, singular parameterisation might offer a solution in the former case, though most likely not so in the latter.

The remaining method — using rational patches — works in both settings. Assuming Gregory patches with cubic boundaries, the VEC is trivially satisfied. Unfortunately, these Gregory patches are equivalent to (bi-)degree 7 rational Béziers, which means that upon splitting a Gregory patch to locally add more detail— resulting in subpatches that are *not* Gregory patches themselves as their boundaries are rational rather than polynomial — we would have to pass on much more data from the CPU to the GPU, which poses a difficulty because of the current limitations on the patch size in the OpenGL drivers. Nevertheless, in cases where local refinement is not required, it offers an easy solution. What is more, it might actually be preferred over the use of Béziers in such cases, as the  $G^1$  mechanics for the latter have a slightly larger region of influence (i.e. moving a control point or tangent handle causes subtle changes in the gradient mesh some distance away from it, and this distance is slightly larger for Béziers compared to Gregory patches).

Another approach would be to use subdivision surfaces. Clearly, the use of approximate schemes such as Catmull–Clark or Loop would not yield satisfactory results as the colours defined at the control points are typically not interpolated by such schemes. Although this indicates that interpolatory schemes could work, note that these might cause overshoots, causing the colour values to go outside the gamut, resulting in flat spots of colour. Figure 6.19 illustrates the two issues.

Still, approximate schemes can be used if the curve network is pre-processed. The



**Figure 6.19:** Sections of subdivision curves approximating (top) and interpolating (bottom) a univariate control net. The approximating scheme (the cubic B-spline subdivision) clearly does not interpolate the colours assigned to the control points, whereas the interpolating scheme (the four-point scheme [DLG87]) overshoots and ventures outside the gamut.

idea is to apply a single step of ternary linear subdivision first, followed by binary Catmull–Clark subdivision [Lie+17]. This ternary step shows similarities to the  $0^\circ$  tangent handles in the Bézier approach we discussed, as it creates groups of three co-linear colour values in the univariate setting and groups of  $2n + 1$  co-planar colour values in the bivariate setting<sup>†</sup>. Using the appropriate limit stencils then ensures the interpolation of the colour specified at the original control points. However, note that in general the geometry is *not* interpolated. Although the ternary step could be tuned to ameliorate this issue, it would also slightly reduce the flexibility of the method.

Local refinement in a subdivision setting is another challenge. Although it is now well-known that the subdivision splines can be refined in a hierarchical fashion (with the optional use of a truncation mechanism), this is not reflected in the local subdivision of the control net. Instead, *control vectors* [KSD15] can be used, an idea that was recently explored in [VK18].

### 6.2.3 Sharp colour transitions

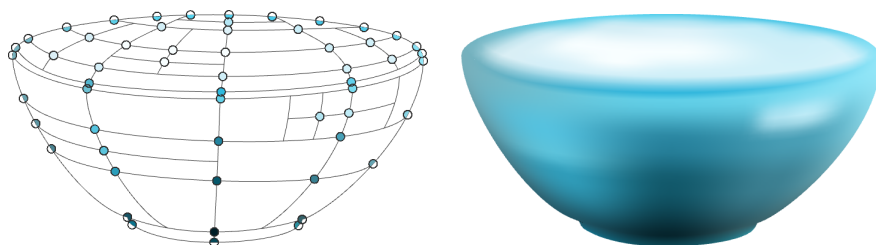
A third — though less severe — shortcoming of the traditional gradient mesh is its inability to facilitate efficient sharp colour transitions. Although there are various workarounds (e.g. collapsing an entire row/column of the mesh, or drawing several gradient

<sup>†</sup> The subdivision setting is more flexible though, as co-linearity in the physical domain suffices, providing some freedom for the placement of the new control points.

meshes on top of each other), they are not exactly user-friendly. An easy solution is to assign colour to individual *sectors* of control points, which is a straightforward feature to implement and intuitive to the user. The next section showcases a couple of examples using this feature.

## 6.2.4 Gallery

Our in-depth look at gradient meshes would not be complete without a couple of (manually designed) examples showcasing our improvements. In Figure 6.20, a gradient mesh using local refinement is shown, which is mostly used to model specular highlights. Adding the branching feature to the mix already allows the design of a much larger variety of shapes, as illustrated in Figure 6.21. Finally, sharp colour transitions facilitate the representation of lifelike objects using only a single gradient mesh, as shown in Figure 6.22.



**Figure 6.20:** A ceramic bowl designed using local refinement, here mostly used to model specular highlights.

## 6.2.5 Colour spaces

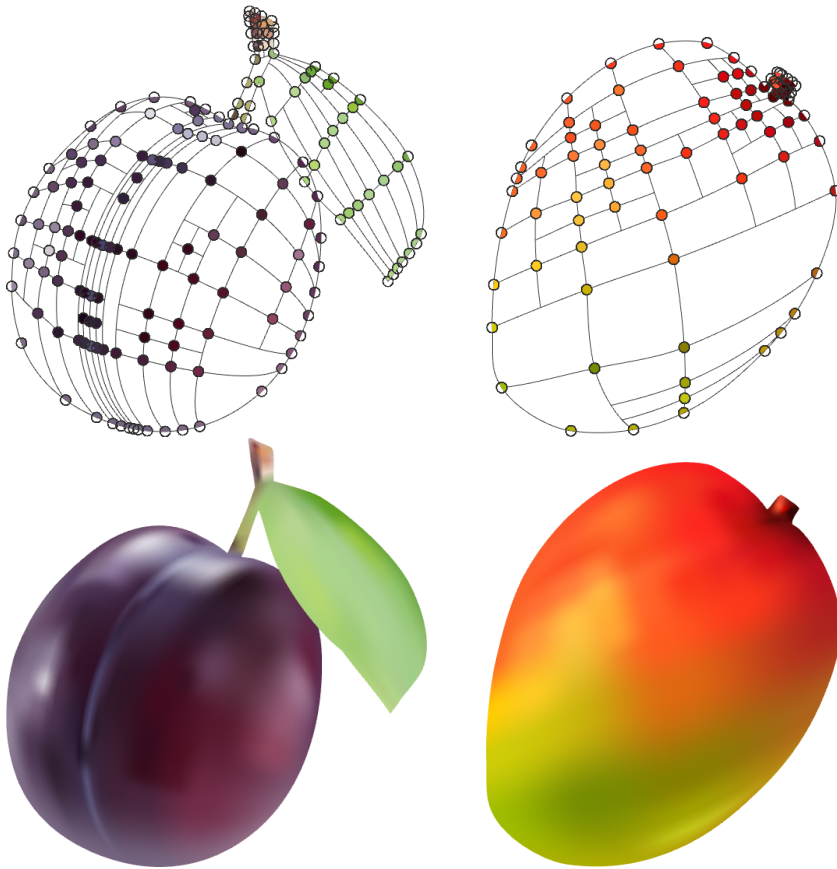
When defining gradient meshes, we assumed the colour space to be sRGB, which makes sense as it is universally known and available. However, there are several alternatives that should be mentioned.

When comparing sRGB to CIEXYZ, CIELAB and CIELUV<sup>†</sup> (see Figure 6.23), dark regions in sRGB space between two different colours can be observed. In contrast, the CIEXYZ space yields overall lighter results. Comparing the remaining spaces — CIELAB and CIELUV, both perceptually uniform — notice that in CIELAB space a purple region occurs between blue and yellow.

Although this very brief comparison hints at CIELUV being the best candidate, in practice the difference between rendering in the default sRGB colour space and CIELUV is hardly noticeable because the colours of neighbouring control points are usually much less extreme than depicted in the example.

---

<sup>†</sup>We do not consider cylindrical spaces such as HSV and HSB, as their periodic nature can result in non-intuitive or ambiguous interpolation of colour.



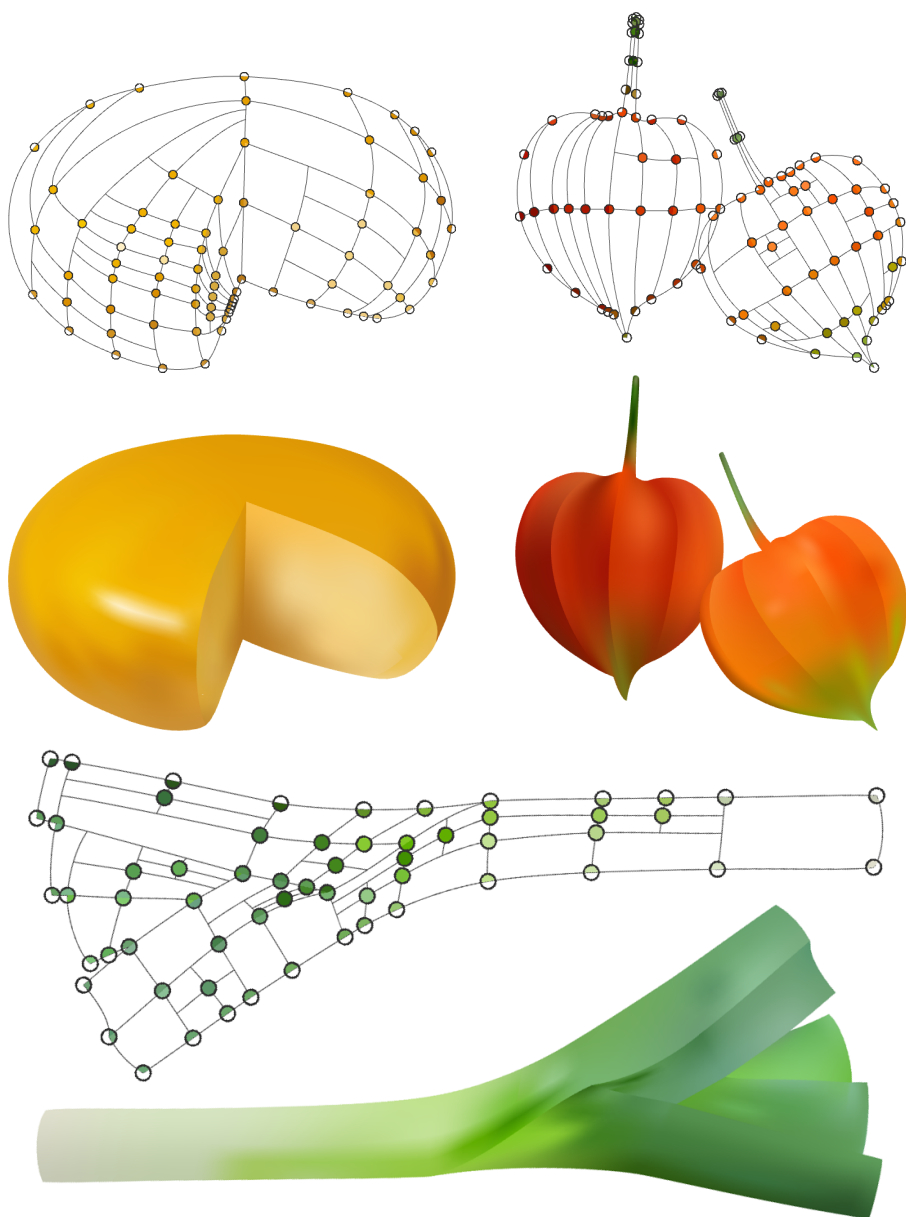
**Figure 6.21:** A variety of fruits modelled using both local refinement and branching. The leaf of the plum is a branch as well, which overlaps the body of the plum.

## 6.2.6 Gradient meshes and the web

Although the gradient mesh is the most widely available primitive for the representation of lifelike images in vector graphics software, it is currently not supported by modern open standards. The upcoming SVG 2.1 aims at including a form of gradient meshes, though unfortunately a partially bicubic Coons patch — which is equivalent to a Ferguson patch when cubic boundary curves are used — was chosen as the basic building block in favour of a fully bicubic patch. As we have shown, the latter is required to handle e.g. exact splitting. Additionally, it provides flexibility regarding the choice of twist-vectors.

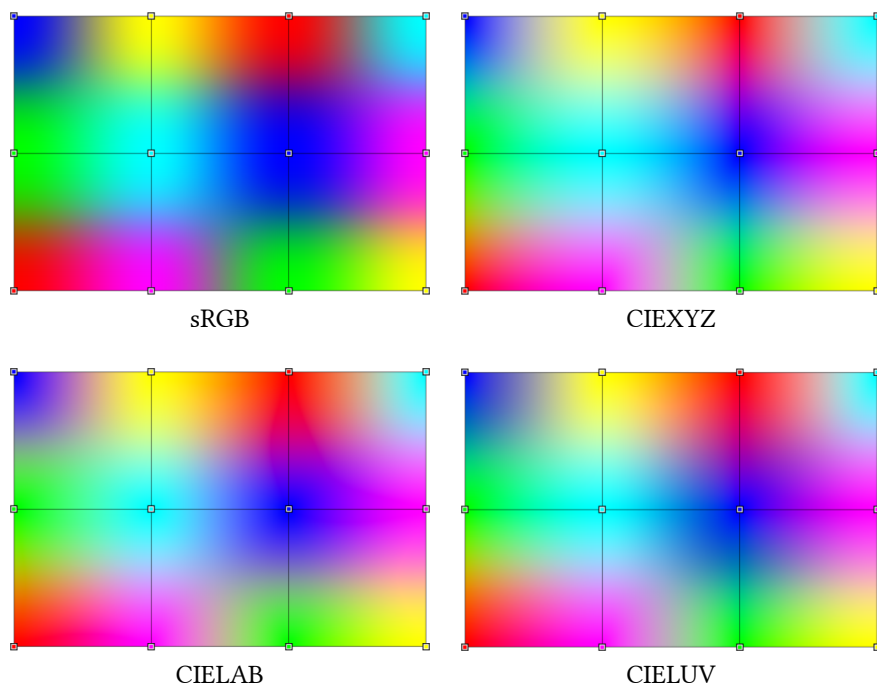
An additional difficulty in establishing an open standard supporting gradient meshes appears to be the lack of interest from web browsers<sup>†</sup>. This is somewhat surprising, as the vectorisation of raster graphics has been steadily increasing in popularity. Moreover, as mentioned earlier, an added benefit of vectorisation is the compression

<sup>†</sup>Based on discussions leading up to and during the Libre Graphics Meeting 2019.



**Figure 6.22:** The ability to assign colour to individual sectors of control points makes gradient meshes much more versatile. All models make use of local refinement. The Chinese lanterns and the leek also use branching, which in the latter case is particularly effective.

it comes with. As example, consider the Mango from Figure 6.21. Saved as a raster image of  $805 \times 916$  pixels in PNG format using maximal compression, it takes up about 170 KB. The locally-refined gradient mesh stored in ASCII format using an OBJ-like



**Figure 6.23:** Renderings of a  $2 \times 3$  gradient mesh interpolated in different colour spaces. Illustrations by Jonathan Hogervorst.

structure takes up approximately 70 KB, which can be brought down to about 20 KB using e.g. gzip file compression.

At the moment of writing, the state-of-the-art open technology on the web is WebGL 2. Unfortunately, it does not support tessellation shaders, which I would argue are an essential component in efficiently rendering gradient meshes. Although tessellation can to some extent be emulated using *instanced rendering* — in our setting the geometry and colour data can be stored as textures (not as vertex attributes<sup>†</sup> or as texture buffers<sup>‡</sup>) — this only covers uniform tessellation and not the required (adaptive) crack-free tessellation. The first version of WebGL’s successor, WebGPU, is therefore eagerly awaited.

## 6.3 Diffusion curves

Approximately ten years after the introduction of the gradient mesh, another primitive with largely the same goals was proposed: the diffusion curve [Orz+08]. The concept is to first draw the outline and inner details of an object using cubic Bézier curves, an

<sup>†</sup>A full bicubic patch requires  $16 \times 5 = 80$  floats, which exceeds the common limit on the number of OpenGL vertex attributes (16) — note that 5 mat4 attributes amount to 20 vec4 attributes.

<sup>‡</sup>Texture buffers are only available as of the latest version of OpenGL ES (3.2). As WebGL 2 is based on ES 3.0, they are not supported.



approach which closely mimics the typical workflow of an artist. Next, piecewise linear colour gradients can be assigned to the curves, optionally different ones on either side. Ultimately, these colour gradients are *diffused* on the canvas such that a smooth colour propagation is obtained.

Diffusion can be captured by a PDE known as the diffusion equation, which is commonly expressed as

$$\frac{\partial u}{\partial t} = c\Delta u + f, \quad (6.5)$$

with  $u$  the unknown (i.e. the red, green or blue colour channel),  $t$  the time,  $c$  the diffusion coefficient and  $f$  a source term. Usually, we are interested in the *steady-state* or time-independent solution for which  $\frac{\partial u}{\partial t} = 0$ . In that case, with the reasonable assumption that  $c$  is constant, the diffusion equation turns into the Poisson equation  $-\Delta u = f$ . Without source term, it further simplifies to the Laplace equation. As we saw in Chapter 4, we require an appropriate set of boundary conditions in order to make the problem well-posed – the presence of a Dirichlet boundary condition at least somewhere on the boundary does the trick.

In the case of traditional diffusion curves, the solution  $u$  is prescribed on all curves (possibly with different values on the two sides of the curves). Assuming Dirichlet boundary conditions everywhere on the boundary of the canvas, the solution to Poisson’s problem is uniquely defined.

If open curves are present in the interior of the canvas, this implies that we cannot simply take the colour values associated with these curves as the solution  $u$  at those locations, as this would most likely violate the solution associated with the boundary value problem (BVP). Instead, the original publication encodes the colours prescribed along open curves as a global source term, which corresponds to the divergence of the colour gradient at the open curves. Alternatively, open curves can be interpreted as holes of infinitesimal area. This allows us to model such open curves as boundaries<sup>†</sup>, which ultimately corresponds to solving Laplace’s problem. We assume this approach throughout the remainder of this section.

### 6.3.1 Solvers

Clearly, in order to solve the process of colour diffusion, we require a numerical method. The original work used a multigrid approach [Orz+08], the details of which we will not discuss here. Instead, we briefly discuss some relevant aspects when using one of the previously discussed numerical methods.

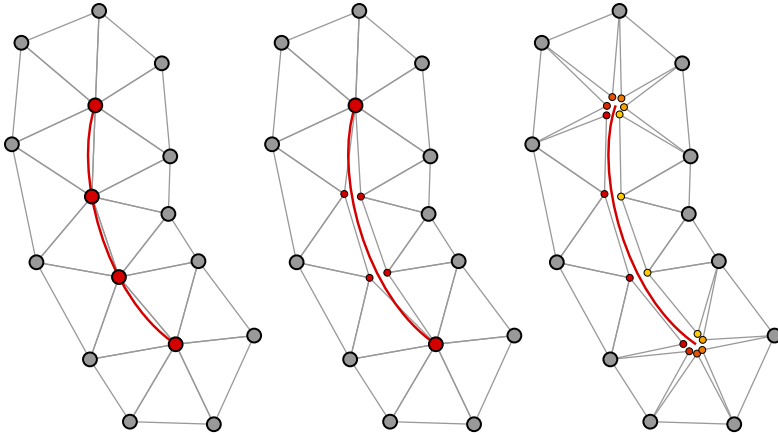
#### Finite element method

In case FEA is used to solve the diffusion process, the first step is to mesh the canvas. This can be done using a process known as *constrained* Delaunay triangulation, which samples the curves at a number of points, connects these samples using edges, and subsequently uses these edges as constraints. This ensures a high-quality triangulation

<sup>†</sup>Strictly spoken, the domain is no longer Lipschitz-continuous in this setting. This can be remedied by offsetting both sides of the curve by e.g. one pixel in the direction of the curve’s normal.

of the domain without triangles crossing the curves. Both linear and higher-order elements can be used — in the latter case, it might be desirable to sample additional points on the curves as additional nodes on the (then curved) edge of the element.

If open curves are present, the mesh has to be ‘cracked open’ along these curves, thereby duplicating nodes. If also the colour profiles are different on the two sides, the end points have to be split to accommodate a transition between the two sides. Figure 6.24 illustrates the two cases.



**Figure 6.24:** Points on an open curve are sampled to construct a triangle mesh (left). In order to model such a curve as a boundary, the mesh is virtually split along it, which duplicates the nodes on the curve (middle). In case of different colour profiles on the two sides, also the end points need to be split (right).

The usual further steps of FEA can now be followed, though care should be taken regarding the integration around the extremities of open curves, as well as possible refinement in these regions. Based on the existing literature, FEA does not seem a common choice in the context of diffusion curves — so far it has been applied in [PJS15], which uses a slightly different approach than the one sketched above.

### Boundary element method

Still assuming the Laplace approach, BEM is a potentially ideal candidate to solve the diffusion problem. The occurrence of open curves can be interpreted as a so-called *screen problem* [SS10], which might be solved using e.g. [Kru00]. Alternatively, we can interpret these ‘cracks’ in the interior as holes in the interior like before.

After solving both the colour values and their flux on the boundary, the interior solution can be obtained by evaluating the boundary integral. Clearly, doing this on a pixel level is rather expensive. Instead, the interior can be meshed in a similar fashion as for FEA, after which the approximation at the element nodes can be evaluated.

There are several works focusing on applying BEM to render the illustrations defined by diffusion curves [Sun+12; Ilb+13; STZ14], one of which leverages the fast multipole method to speed up the process [STZ14].

## Ray-tracing

In addition to the multigrid method, FEM or BEM, a fourth technique has been used repeatedly, though unlike the former three techniques, it is not based on the diffusion equation. Instead, the ray-tracing approach is based on the *render equation*. Starting again from a triangular mesh as described before, it shoots a number of rays from every vertex in the triangulation, checks for intersection of those with the curves, and typically adds up the obtained values weighed by the inverse of the distance of the hits. Rays that do not intersect can be said to intersect with a black curve at infinite distance.

In a preliminary implementation of the concept, we can deform a number of (open) curves using the same linear colour distribution on both sides. An example containing multiple such curves, corresponding triangulation and rendered result are visualised in Figure 6.25.

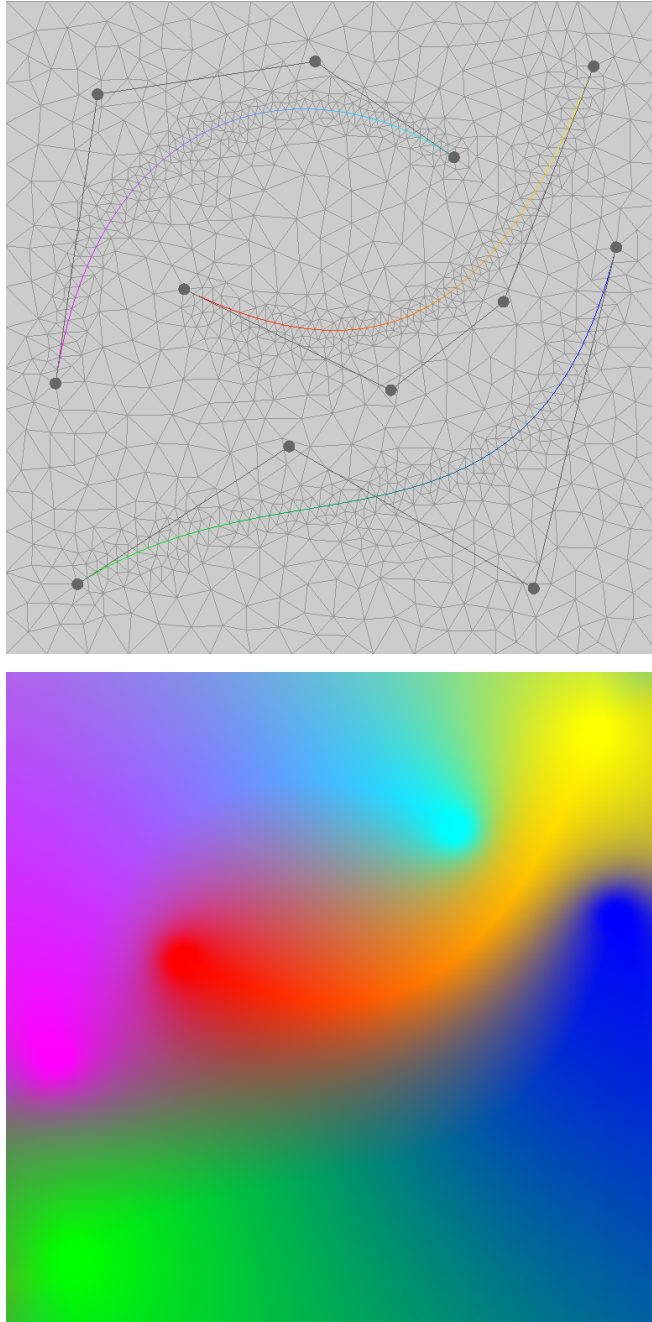
For literature using ray-tracing in the context of diffusion curves, we refer to [BLW11; Pan+12; PJS15].

### 6.3.2 Improvements

Soon after the publication of the original work, various changes and improvements were proposed. One of these is to replace the Laplace operator by the *biharmonic* operator, typically yielding smoother results [FSH11; Jes16]. Other contributions include the concepts of barrier curves [Bez+10] and diffusion points [FSH11]. The former allows to have curves without colour profiles specified on one or both sides, which act as barriers on the canvas (i.e. to prevent the colour from spreading in certain areas). Diffusion points can be interpreted as point sources which can be placed in the interior of the canvas. Similar to open curves, these can be modelled as (circular) punctures in the canvas. In the ray-tracing approach, these can be added to the list of intersection tests as small discs — in fact, they are included in the example shown in Figure 6.25 at the endpoints of the curves. Barrier curves and diffusion points facilitate a workflow that is arguably more intuitive than the original process, which requires colour profiles for all curves. This seems to be the mechanics behind the recent introduction of *freeform gradients* in ADOBE ILLUSTRATOR.

Apart from the creation of resolution-independent artwork, applications of diffusion curves include the use of the 2D results as textures for 3D objects [Sun+12]. There is also recent work using diffusion curves to vectorise raster graphics [ZDZ17].

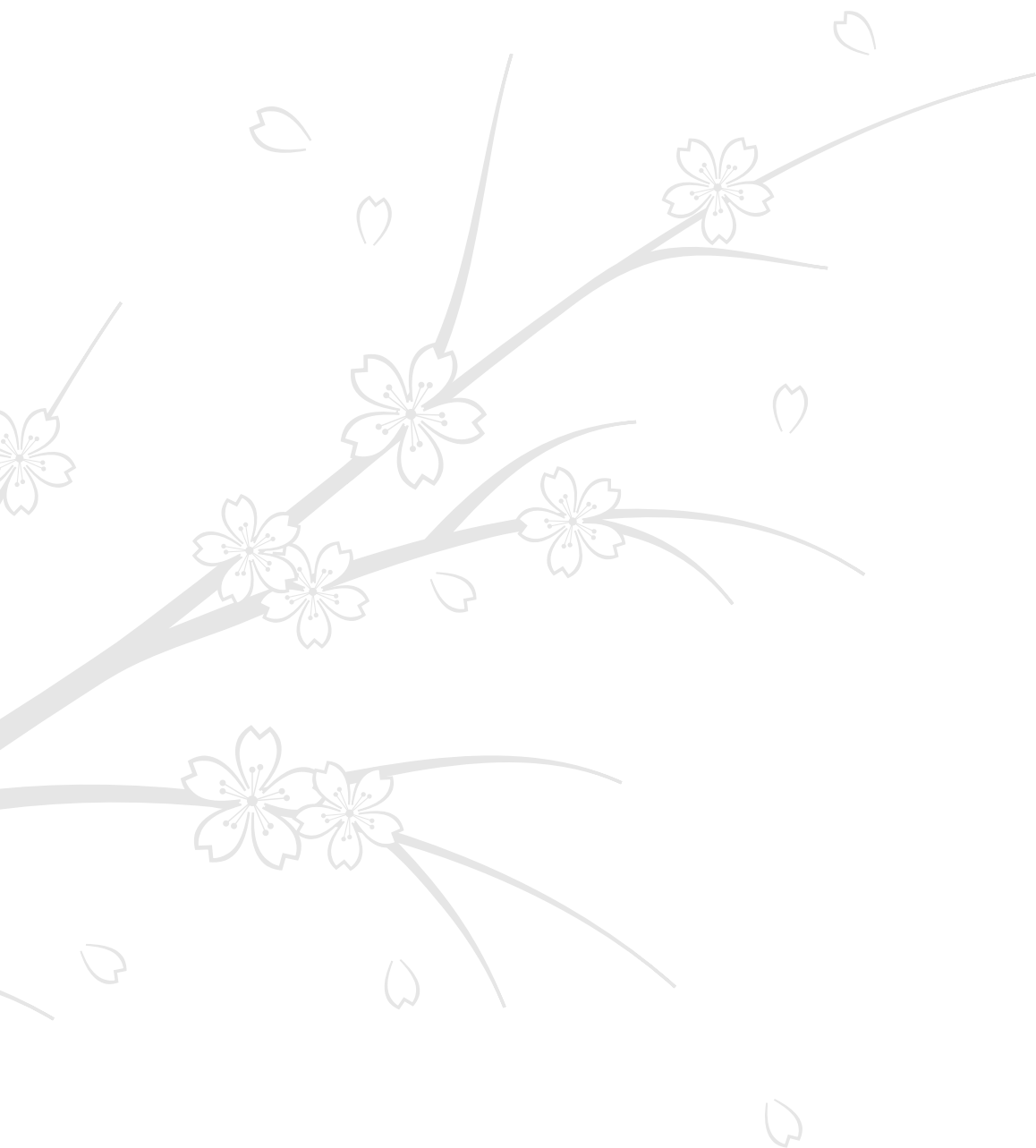
In the brief discussion above, we did not touch upon the possibility of intersecting curves. Many papers include this as a special case, and although it is of academic interest to see how a method performs in such a setting, in practice such a scenario rarely occurs. I would argue that the same holds for open curves with different colour profiles specified on its two sides.



**Figure 6.25:** Three open cubic Bézier curves with linear colour profiles specified along them. They are each sampled at a number of points, creating the edges constraining the triangulation (top). Using ray-tracing, with 128 uniformly distributed rays for every vertex in the triangulation, the diffusion curve image is rendered using linear interpolation over the triangular elements (bottom).

## 7 | Conclusion and future work





In this last chapter we look back on the discussed concepts and contributions. Furthermore, we share some thoughts on interesting directions for future research.

## 7.1 Conclusion

It can be difficult to explain to outsiders why your PhD project takes about four years to complete. Perhaps I was even among them at the start of my PhD, when there was still so much time to delve into the literature, become more familiar with programming languages and attend courses on a variety of ‘soft’ skills, just to name a few things. It almost goes without saying that I have learned and further developed quite a diverse range of skills these past years.

As I already mentioned in the introduction, the stage of drafting and writing this dissertation has been one of the most enjoyable experiences of my PhD. Surely enough, I have cursed myself more than once for rewriting almost everything from scratch instead of modifying the publications and adding them as chapters one by one. Ultimately, I am glad I went with this specific approach and hope that the result will be useful to others interested in one or more of the concepts discussed, illustrated and occasionally proved.

In particular, I hope that the chapter on splines can offer some insights that can be hard to distil from the more mathematically oriented texts. Béziérs are so often introduced by simply stating an expression involving the Bernstein polynomials, which I am convinced does not add much to the intuitive understanding of most readers. Defining them using de Casteljau’s approach instead seems the natural approach to me, which at the same time makes it inevitable to appreciate the elegance of these curves. Similar things can be said about the introduction of B-splines through the Cox-de Boor recurrence relation — defining B-splines as a natural way of connecting Bézier curves with built-in continuity makes much more sense to me. I could go on, mentioning that the repeated convolution of the unit pulse with itself directly shows that the projection of certain polytopes leads to the same results, but I trust that my point is clear. Nevertheless, the content could still be improved and expanded — as mentioned, I will continue towards this goal and plan to provide separate notes on [SplinesForEngineers.com](https://splinesforengineers.com).

The next chapter, focusing on splines of arbitrary manifold topology, provided a pragmatic overview of the  $G^1$  interpolation of cubic curve networks which I am currently using for a follow-up paper regarding flexible gradient meshes. In addition, it allowed me to elaborate on various aspects of subdivision surfaces. The section discussing the subdivision scheme based on half-box splines covers the first publication discussed in this dissertation (which is chronologically speaking actually the latest publication). I think it is a very suitable scheme to explain the characteristics of subdivision, all the way from connecting the control points to form a control net (which is indeed an artificial process) to the occurrence of ineffective eigenvectors. The peculiar three-valent meshes lend themselves well for architectural and artistic purposes.

Writing about numerical methods felt a little like time travel (or at least, what I imagine it to feel like). It was nice to highlight the different parts of FEM, something which I did not get to do in my MSc thesis. Focusing on improving quadrature for subdivision splines, including the short internship with Michael Bartoň at BCAM, has been a pleasant journey, and one that is still ongoing — a second publication is expected in the near future. It is also a somewhat personal topic. Back when I first presented about subdivision-based IgA, an often-heard remark was that subdivision did not pass the patch test. Of course, by the very definition of isoparametric elements, this is a strange

comment — it is purely a matter of efficiency (and as our work shows, considerable improvement can be made compared to the default approach). In addition to FEM, I had to include some notes on BEM, which is an elegant method often overlooked, even though it can be (much) more efficient than FEM in some settings. It turns up again in the context of colour propagation in vector graphics. The chapter also includes a brief discussion on spline-enhanced FEM, which together with former colleagues from Leuven I extended to subdivision splines.

Tessellation and shading were discussed next, and are typical examples of computer graphics. I remember my first efforts at implementing a functional OpenGL tessellation shader, something which seems completely trivial now. Extending it to multi-sided patches is a challenging generalisation. Shading is a topic with many facets — pun intended — of which only a few were discussed. In addition to our incremental work on subdivision shading, there are many remaining applications. Regarding graphics APIs, the future is going to be exciting, with Vulkan (eventually also including OpenCL for GPGPU) and WebGPU.

The last core chapter is an exciting mixture of mathematics, engineering and computer science with the occasional pinch of art. Colour propagation in vector graphics is a very nice and visual topic to work on. Unlike some of the concepts in other chapters, it is straightforward to explain on a basic level. This is perhaps also the reason for the more unusual forms of dissemination, including a rendering (using the improved gradient mesh primitive) on a promotional MSc banner of the University of Groningen, a short article in the faculty magazine as well as an article in the German and US Linux Magazines [Bar19a; Bar19b]. Diffusion curves form an interesting alternative to gradient meshes and are arguably much more intuitive. Finally, vectorisation is likely going to be more important in the near future, especially once these advanced primitives (which might sound somewhat contradictory to the outsider) are supported by the main web browsers.

In addition to (literature) research, programming and writing, there were various other aspects of academia that I spent my time on. These include teaching (in my case mostly putting together seminars, assignments and implementing functional cross-platform frameworks for the students to work with) and attending conferences. Though most conferences in our field are only attended by academics and researchers from companies, at others such as the Blender Conference or the Libre Graphics Meeting the developers and artists have the upper hand. This was an interesting contrast and something that I think should be encouraged, especially in the area of computer graphics — researchers, developers and artists can learn a lot from each other.

## 7.2 Future work

The virtual shelf of future projects has been replenished rather than emptied throughout my PhD. For some of those that are related to the matters discussed here, I include a brief description below.

- Just like B-splines can be constructed by imposing fixed continuity between connecting functional Bézier segments (and evaluated using a de Casteljau-like approach), it should be possible to do the same thing for (half-)box splines and per-



haps other families of splines. Although blossoming might not generalise to all of these, geometric constructions for the evaluation of patches must be available based on their coordinate-free definition (i.e. only using affine combinations). This might lead to new families of splines and construction methods, or at least to insights in the connection between different forms of splines.

- Subdivision surfaces have surprisingly much in common with the concepts of reptiles and self-tessellating tile sets. That is, the two-scale relation for e.g. a box spline can be interpreted as a generalisation of reptiles. In both cases, an original function is composed of scaled and shifted copies of itself. Whereas for box splines these copies overlap (they have to), for reptiles they do not — this can be explained by interpreting reptiles as constant functions on polygonal domains (which is in fact the first step in defining the mask of a box spline by means of discrete convolution).

Going a step further, we have seen that a set of subdivision splines can be composed of scaled and shifted copies of the same set. This turns out to be analogous to the notion of self-tessellating tile sets — given a set of tiles (some of which may actually be reptiles), these can be tiled using scaled and shifted copies of the same tile set.

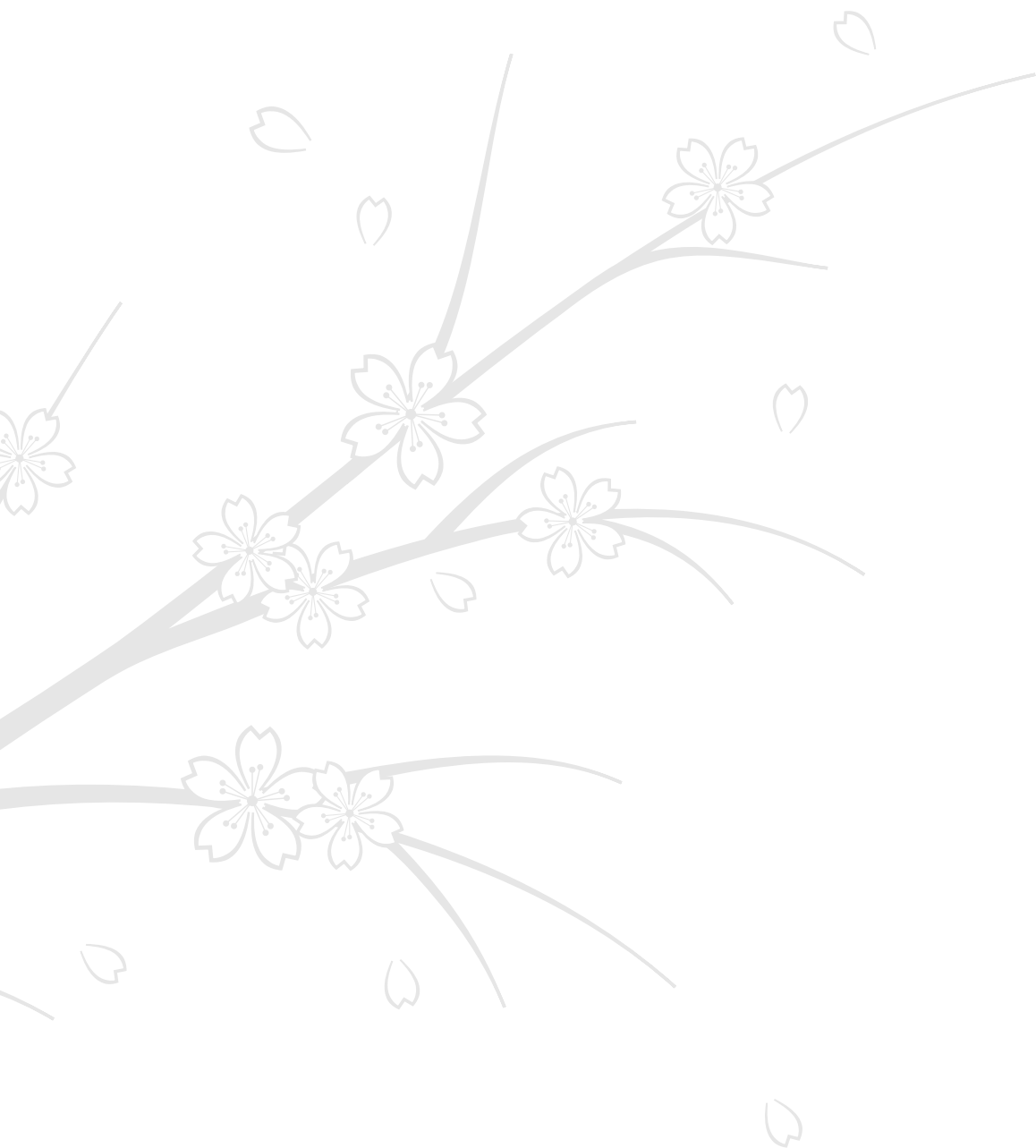
Not only is this an interesting link to a topic in recreational mathematics, it also provides a different angle of looking at subdivision and perhaps a framework for further discovery of new schemes.

- An entertaining application of subdivision surfaces could be their use in classic games, including game of life, snake, mine sweeper, pacman and so on — these can be generalised to other types of grids and indeed to meshes of arbitrary manifold topology. The control net defining the subdivision surface could be projected onto the limit surface to visualise the grid.
- In addition to completing optimised quadrature rules for subdivision-based IgA, it would be interesting to take a look at approximated Catmull–Clark surfaces (e.g. using Gregory patches) in a (volumetric) IgA context. This might be interesting for physics engines using such objects in games or animated movies.
- The work on using composite  $G^1$  Bézier patches for smooth colour propagation in the context of gradient meshes (combined with refinement at arbitrary parameter values) probably closes the chapter on manual creation using the primitive — I do not see how it could be generalised any further without losing user-friendliness. Notwithstanding, support for the technology is still lacking, especially so in web browsers. My recent implementation of a gradient mesh renderer in WebGL 2 might be helpful in getting the (modified) gradient meshes in the SVG 2.1 proposal accepted.



# Appendices





# A | Local refinement

Local refinement of splines is crucial in both spline-based numerical simulation (see Chapter 4) and modelling. In Chapter 2 we already looked into minimal refinement and the two-scale relation. Building on the latter, we discuss various refinement methods and illustrate them in the univariate setting.

## A.1 Hierarchical B-splines

Recall that a knot-vector  $\Xi = [0, 1, 2, \dots, d + n] \subset \mathbb{Z}$  defines a sequence of  $n$  uniform B-splines of degree  $d$ , i.e.  $\mathcal{M}^d(t - k)$  with  $k \in [0, n - 1] \subset \mathbb{Z}$ . Copying this knot-vector and applying midpoint refinement to it provides us with a knot-vector  $\Xi_r \subset \mathbb{Z}/2$ , which defines a sequence of  $2n + d$  uniform B-splines  $\mathcal{M}^d(2t - l)$  with  $l \in [0, 2n + d - 1] \subset \mathbb{Z}$ . Further dyadic refinement results in uniform B-splines with knots in  $\mathbb{Z}/4, \mathbb{Z}/8$ , and so on.

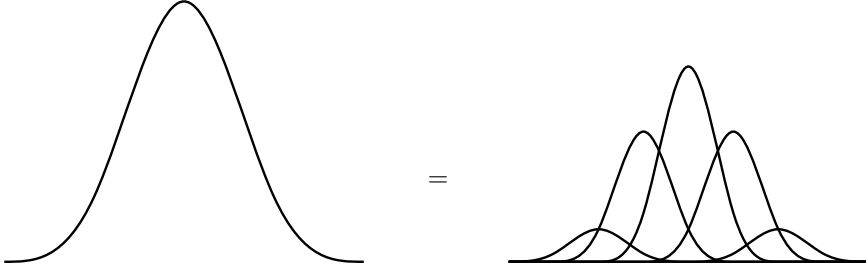
The idea of hierarchical B-splines [FB88] is to iteratively combine uniform B-splines defined on increasingly finer grids  $\mathbb{Z}/2^m$ , with  $m \in \mathbb{N}$ , in a hierarchical fashion. That is, given our B-splines  $\mathcal{M}^d(t - k)$  spanning a space  $V$ , one or more B-splines  $\mathcal{M}^d(2t - l)$  could be added to  $V$ . However, in order to prevent linear dependencies, one or more original B-splines  $\mathcal{M}^d(t - k)$  might have to be removed from  $V$ . After all, recall that according to the two-scale relation, a uniform B-spline can be composed of dilated and shifted copies of itself. Taking the uniform cubic B-spline  $\mathcal{M}^3(t)$  as example, we have

$$\mathcal{M}^3(t) = \frac{1}{8} \left( \mathcal{M}^3(2t) + 4\mathcal{M}^3(2t - 1) + 6\mathcal{M}^3(2t - 2) + 4\mathcal{M}^3(2t - 3) + \mathcal{M}^3(2t - 4) \right), \quad (\text{A.1})$$

which is illustrated in Figure A.1.

Clearly, if the union of the supports of added B-splines  $\mathcal{M}^d(2t - l)$  contains the support of an original B-spline  $\mathcal{M}^d(t - k)$ , it should be removed from the space [Kra97]. Original B-splines whose support does not meet this condition should *not* be removed, as doing so would violate the hierarchical nature of the refinement.

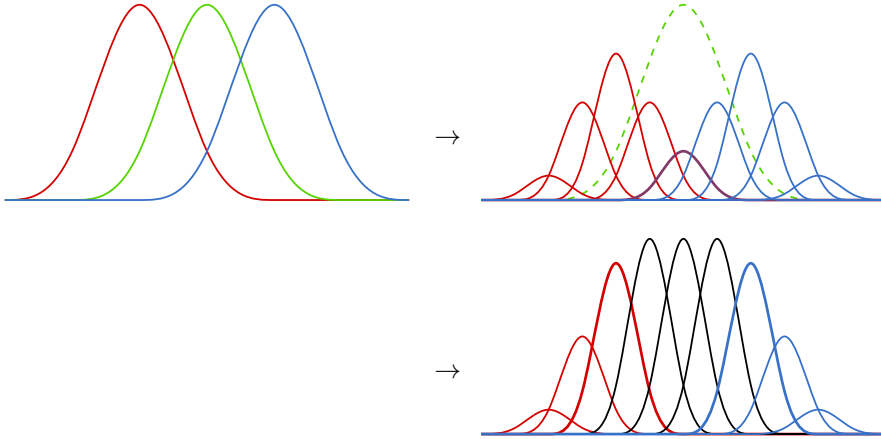
A drawback of this approach is that the refined basis does not partition unity any more. A partial solution is to replace selected original B-splines according to the two-scale relation. Figure A.2 shows an example, which replaces the original B-splines  $\mathcal{M}^3(t)$  and  $\mathcal{M}^3(t - 2)$  by appropriately scaled B-splines  $\mathcal{M}^3(2t - l)$ . Note that a



**Figure A.1:** The two-scale relation illustrated for the uniform cubic B-spline  $\mathcal{M}^3(t)$  cf. (A.1).

scaled version of  $\mathcal{M}^3(2t - 4)$  is added twice — these two copies can be combined by simply adding the scaling coefficients.

In this example, the union of the supports of the added B-splines  $\mathcal{M}^3(2t - l)$  contains the support of the original  $\mathcal{M}^3(t - 1)$ , which should therefore also be replaced according to the two-scale relation.



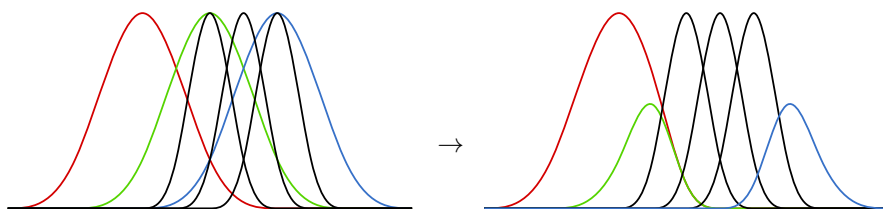
**Figure A.2:** Replacing original B-splines  $\mathcal{M}^3(t - k)$  (left) according to the two-scale relation (right).

Note that in the example, the B-splines  $\mathcal{M}^3(2t - 3)$ ,  $\mathcal{M}^3(2t - 4)$  and  $\mathcal{M}^3(2t - 5)$  have corresponding scaling coefficients of 1.0. This implies that they can be associated with new control points that are obtained by applying the stencils  $[1, 6, 1]/8$  and  $[4, 4]/8$  from Section 3.2 to the original control points. The remaining  $\mathcal{M}^3(2t - l)$  are associated with original control points. The introduction of new control points is especially helpful in the context of modelling, as it gives a designer the possibility to (locally) change the shape of a curve or surface.

The notion of hierarchical refinement is often used in spline-based numerical methods [Vuo+11; Kur+13] and clearly generalises to the tensor-product setting, but also to box splines [Kan+17] and in fact subdivision splines in general [Wei+15; ZJK16].

## A.2 Truncated hierarchical B-splines

Further improvements can be made to hierarchical B-splines by introducing a truncation mechanism [GJS12]. The idea is relatively straightforward — in this scenario, the B-splines  $\mathcal{M}^3(2t-l)$  are always inserted with a scaling factor of 1.0, and are subtracted from *all* original B-splines  $\mathcal{M}^3(t-k)$  containing traces of them (using the appropriate coefficients from the two-scale relation)<sup>†</sup>. As a result of truncation, the support of the original B-splines may decrease; Figure A.3 shows an example.



**Figure A.3:** Introduction of new B-splines  $\mathcal{M}^3(2t-l)$  (left), followed by truncation of the original B-splines  $\mathcal{M}^3(t-k)$  (right).

This construction, which is referred to as *truncated* hierarchical B-splines (THB-splines), clearly maintains the partition of unity of the basis, as well as its linear independence. In addition to its application in numerical methods, it can be used as a refinement strategy for modelling — each newly introduced B-spline  $\mathcal{M}^3(2t-l)$  can be associated with a new control point.

<sup>†</sup>Note that this can result in truncated B-splines with multiple maxima.

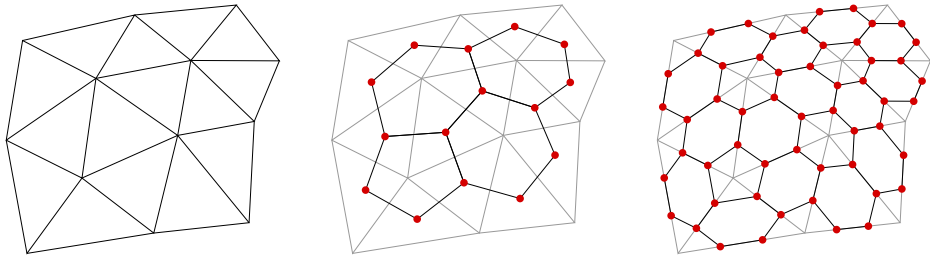
# B | Meshes

Various types and aspects of meshes have been mentioned throughout this dissertation. In this appendix, we consider different approaches to generate three-valent meshes, which can be used as input for the subdivision scheme discussed in Section 3.2.1. Furthermore, we take a short look at the data structure used to implement the cubic curve networks introduced in Section 3.1.1.

## B.1 Generating three-valent meshes

Given a triangle mesh, we consider its *geometric dual*, which generates new vertices at the vertex centroids of the triangles and subsequently connects these over the triangle edges. The result is a three-valent mesh; see Figure B.1.

A similar approach consists of trisecting the edges of the triangle mesh, connecting the resulting vertices around the original vertices and adding edges by connecting the pairs of new vertices on each original edge. The method is demonstrated in Figure B.1; topologically, it is dual to a  $\sqrt{3}$  refinement step [Kob00].



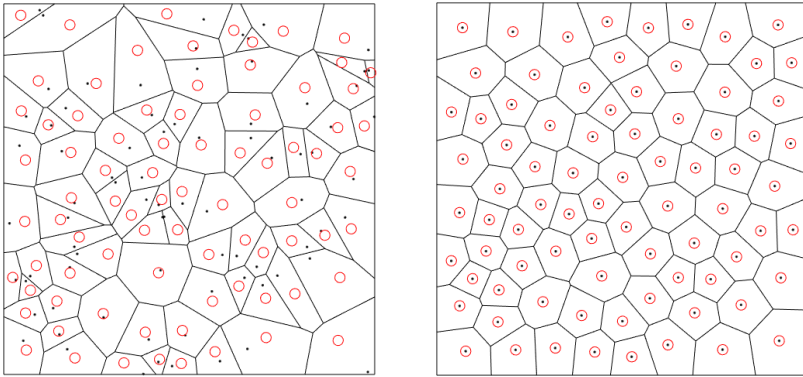
**Figure B.1:** A triangle mesh (left), its geometrical dual (middle) and its trisection (right), both resulting in three-valent meshes.

In addition to the geometric dual, the notion of the *tangent* dual can be used to generate three-valent meshes [Wan+08].

Another approach aimed at constructing three-valent meshes is facilitated by means of centroidal Voronoi tessellations (CVTs) [DFG99; Wan+15]. We explain the concept of CVTs through the use of Lloyd’s algorithm [Llo82]. To start with, a sequence of *generators* (which can be either provided or generated randomly) is used to define a



Voronoi tessellation on a surface<sup>†</sup>. Next, the (vertex) centroids of the resulting cells are computed, after which the generators are moved to coincide with these centroids. By repeating this process, the Voronoi tessellation converges to a *centroidal* Voronoi tessellation, which is characterised by coinciding generators and cell centroids. Figure B.2 illustrates the initial and final stages.



**Figure B.2:** Initial generators (black dots), resulting Voronoi cells and their vertex centroids (red circles) in a bounded domain (left). The eventual CVT is obtained through the use of Lloyd’s algorithm (right).

Although Lloyd’s algorithm is straightforward to implement, it converges rather slowly; faster methods are available [Liu+09].

## B.2 A data structure for curve networks

As mentioned in Section 3.1.1, the starting point of a curve network is a quad-triangle mesh of arbitrary topology. Such structures are commonly stored using a half-edge data structure [Män88] or winged-edge data structure [Bau72]. For non-manifold meshes, there is the radial-edge data structure [Wei88]<sup>‡</sup>.

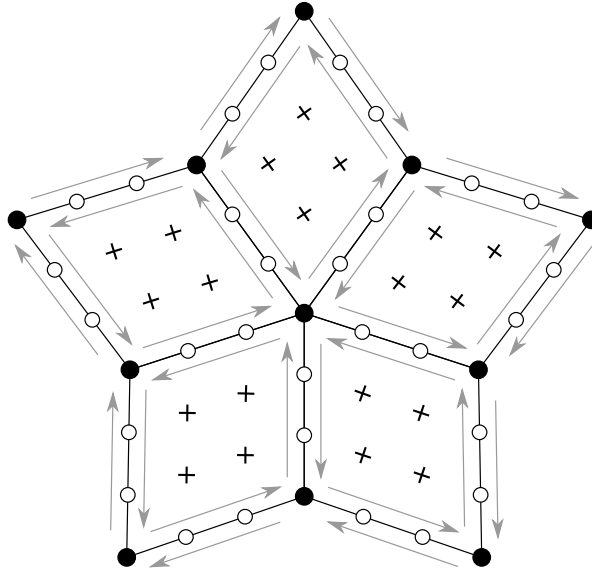
In addition to the control points, a curve network contains tangent handles and interior control points (e.g. twist-vectors) which need to be stored in an efficient manner. I chose to add these to a standard half-edge data structure; see Figure B.3. As tangent handles are shared between adjacent patches, these are made globally accessible through pointers or indices that are stored as attributes of the half-edges. In contrast, twist-vectors belong only to a single patch and can be stored directly as an attribute of the half-edges.

The above only considers geometrical data. In case of a gradient mesh (see Section 6.2), we also need to store colour data. The colour at the control points could be stored as an attribute of those, though this would complicate the support for sharp colour transitions. Instead, we choose to store the colour at the control points and

<sup>†</sup>This includes non-planar surfaces such as triangle- or quad meshes in  $\mathbb{R}^3$ .

<sup>‡</sup>BLENDER’s Bmesh is based on the radial-edge data structure and allows for efficient implementations using Python or C++.

tangent handles, as well as the colour at the interior control points, as attributes of the half-edges. This facilitates the assignment of colour to a single sector of a control point.



**Figure B.3:** An augmented half-edge data structure (half-edges in grey) containing tangent-handles (open circles) and twist-vectors (plus symbols).

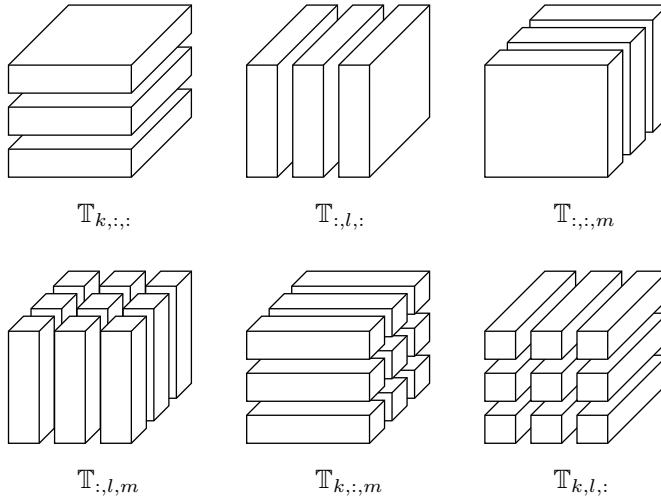
The introduction of T-sections into a half-edge structure can be handled in different ways as briefly discussed in Section 6.2.1. The fundamental choice to make here is whether the half-edge at the stem-side of a T-section should be split or not. If so, the *opposite* property of the half-edges involved is no longer fully symmetrical, which requires consistency throughout the assignment. If the decision is taken *not* to split the half-edges involved, the T-sections become instrumental in traversing the mesh.

# C | Tensors

In order to generalise the approach for the exact integration of products of (derivatives of) subdivision splines described in Section 4.2.1 to triple- and higher-order products, we have to switch to tensor notation. In this appendix we consider triple products of subdivision splines, which requires the use of third-order tensors, also referred to as 3D matrices.

## C.1 Terminology and notation

We adapt the terminology and notation from [KB09], see Figure C.1. The elements of a third-order tensor  $\mathbb{T}$  can be indexed using three numbers  $k, l, m \in \mathbb{N}$ . Keeping one of the indices constant while the other two can take on any nonnegative value within their range results in *slices* (also referred to as layers or pages). Keeping two indices constant while the third one can vary yields *fibers*.



**Figure C.1:** Horizontal, lateral and frontal slices of a third-order tensor (top) and its column, row and tube fibers (bottom).

The product of a third-order tensor and a *matrix* is computed by multiplying each

fiber of the tensor with the matrix, which ultimately results in another third-order tensor. As it can be ambiguous which fiber orientation should be selected (e.g. in case of a tensor with the same range in multiple dimensions), this operation is expressed as  $\times_n$ , with  $n \in \{1, 2, 3\}$  indicating the orientation. Likewise, the product of a third-order tensor and a *vector* is computed by taking the inner product of each fiber with the vector, which ultimately yields a 2D matrix. For this type of product, the notation  $\bar{\times}_n$  is used.

## C.2 Triple products of subdivision splines

The integration of a triple product of subdivision splines over  $\Omega$  requires a  $16 \times 16 \times 16$  tensor  $\mathbb{M}$  containing the triple products of the segments of uniform bicubic B-splines. As these are of bidegree 9, we choose to use a  $5 \times 5$  Gauss–Legendre rule to integrate them exactly. Next, we define the shorthand notation

$$\mathbb{C}_n^k = (X_n^k \bar{\mathcal{A}}_n^S V_n)^T \times_1 \left( \left[ \sum_{G=1}^{25} w_G \mathbb{M}(s_G, t_G) \right] \times_2 (X_n^k \bar{\mathcal{A}}_n^S V_n) \right) \times_3 (X_n^k \bar{\mathcal{A}}_n^S V_n), \quad (\text{C.1})$$

which then allows us to follow the same steps as in (4.31) and (4.32). With  $\Gamma_n = \frac{1}{\sqrt[3]{4}} \Lambda_n$ , this yields

$$[\mathbb{S}_n^k]_{abc} = \frac{[\mathbb{C}_n^k]_{abc}}{1 - [\Gamma_n]_{aa} [\Gamma_n]_{bb} [\Gamma_n]_{cc}}, \quad (\text{C.2})$$

so that

$$\int_{\Omega} \varphi_{n,a}(u, v) \varphi_{n,b}(u, v) \varphi_{n,c}(u, v) d\Omega = \frac{1}{4} z_{n,a}^T \left( \left( \sum_{k=1}^3 \mathbb{S}_n^k \right) \bar{\times}_2 z_{n,b} \right) z_{n,c}. \quad (\text{C.3})$$

Using higher-order tensors, the same approach can be followed to compute e.g. centroids and moments of inertia of subdivision surfaces.

# Bibliography

- [Ado97] Adobe Developers Association. *Smooth Shading*. Tech. rep. Adobe Systems Incorporated, 1997 (cit. on p. 158).
- [Ado98] Adobe Systems Incorporated. *Adobe Illustrator 8.0 Classroom in a Book*. Adobe Press, 1998 (cit. on p. 158).
- [Ado99] Adobe Systems Incorporated. *PostScript language reference*. Third edition. Addison-Wesley, 1999 (cit. on p. 158).
- [AO11] Mark Ainsworth and J Tinsley Oden. *A posteriori error estimation in finite element analysis*. John Wiley & Sons, 2011 (cit. on p. 99).
- [AS02] Ergun Akleman and Vinod Srinivasan. “Honeycomb subdivision”. In: *Proc. 17th Int. Symp. Computer & Information Sciences*. 2002, pp. 137–141 (cit. on p. 63).
- [Akl+04] Ergun Akleman, Vinod Srinivasan, Zeki Melek and Paul Edmundson. “Semi-regular pentagonal subdivisions”. In: *Proceedings Shape Modeling Applications*. 2004, pp. 110–118 (cit. on p. 63).
- [AB08] Marc Alexa and Tamy Boubekeur. “Subdivision Shading”. In: *ACM Transactions on Graphics* 27.5 (2008), 142:1–142:4 (cit. on p. 152).
- [AS10] Lars-Erik Andersson and Neil Frederick Stewart. *Introduction to the mathematics of subdivision surfaces*. SIAM, 2010 (cit. on p. 61).
- [ADS06] Ursula H Augsdörfer, Neil A Dodgson and Malcolm A Sabin. “Tuning subdivision by minimising Gaussian curvature variation near extraordinary vertices”. In: *Computer Graphics Forum* 25.3 (2006), pp. 263–272 (cit. on pp. 78, 116).
- [Aur+12] F. Auricchio, F. Calabrò, T. J. R. Hughes, A. Reali and G. Sangalli. “A Simple Algorithm for Obtaining Nearly Optimal Quadrature Rules for NURBS-based Isogeometric Analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 249.1 (2012), pp. 15–27 (cit. on p. 109).
- [Bab73a] Ivo Babuška. “The finite element method with Lagrangian multipliers”. In: *Numerische Mathematik* 20.3 (1973), pp. 179–192 (cit. on p. 101).
- [Bab73b] Ivo Babuška. “The finite element method with penalty”. In: *Mathematics of computation* 27.122 (1973), pp. 221–228 (cit. on p. 101).

- [Baj+02] Chandrajit Bajaj, Scott Schaefer, Joe Warren and Guoliang Xu. “A subdivision scheme for hexahedral meshes”. In: *The visual computer* 18.5 (2002), pp. 343–356 (cit. on p. 126).
- [BXW02] Chandrajit Bajaj, Guoliang Xu and Joe Warren. “Acoustic scattering on arbitrary manifold surfaces”. In: *Proceedings Geometric Modeling and Processing*. IEEE. 2002, pp. 73–82 (cit. on p. 132).
- [Bak01] Almaz Bakenov. “T-Splines: Tensor Product B-spline Surfaces with T-Junctions”. MA thesis. Brigham Young University, 2001 (cit. on p. 84).
- [BBK18a] Jelle Bakker, Pieter J Barendrecht and Jiří Kosinka. “Smooth Blended Subdivision Shading.” In: *Eurographics (Short Papers)*. 2018, pp. 37–40 (cit. on pp. 138, 152).
- [BS86] Alan A Ball and David JT Storry. “A matrix approach to the analysis of recursively generated B-spline surfaces”. In: *Computer-Aided Design* 18.8 (1986), pp. 437–442 (cit. on pp. 66, 71).
- [BS88] Alan A Ball and David JT Storry. “Conditions for tangent plane continuity over recursively generated B-spline surfaces”. In: *ACM Transactions on Graphics* 7.2 (1988), pp. 83–102 (cit. on p. 66).
- [BS90] Alan A Ball and David JT Storry. “An investigation of curvature variations over recursively generated B-spline surfaces”. In: *ACM Transactions on Graphics* 9.4 (1990), pp. 424–437 (cit. on p. 66).
- [Ban+15] Kosala Bandara, Fehmi Cirak, Günther Of, Olaf Steinbach and Jan Zapletal. “Boundary element based multiresolution shape optimisation in electrostatics”. In: *Journal of computational physics* 297 (2015), pp. 584–598 (cit. on p. 132).
- [Bar12a] Pieter J Barendrecht. *Implementing an IgA Compatible User-Element for MSC Marc*. Philips Internship Report. Eindhoven University of Technology, 2012 (cit. on p. 101).
- [Bar12b] Pieter J Barendrecht. *IsoGeometric Analysis with Subdivision Surfaces*. Tech. rep. Eindhoven University of Technology, 2012 (cit. on p. 103).
- [Bar13] Pieter J Barendrecht. “Isogeometric analysis for subdivision surfaces”. MA thesis. Eindhoven University of Technology, 2013 (cit. on pp. 63, 103, 109).
- [Bar19a] Pieter J Barendrecht. “Realistischere Vektorgrafiken”. In: *Linux Magazin* 9 (2019). Translated from English to German by Jens-Christoph Brendel, pp. 24–27 (cit. on p. 185).
- [Bar19b] Pieter J Barendrecht. “Photorealistic images with vector graphics”. In: *Linux Magazine* 229 (2019). To appear (cit. on p. 185).
- [BBK18b] Pieter J Barendrecht, Michael Bartoň and Jiří Kosinka. “Efficient quadrature rules for subdivision surfaces in isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 340 (2018), pp. 1–23 (cit. on pp. 86, 112).

- [Bar+18] Pieter J Barendrecht, Martijn Luinstra, Jonathan Hogervorst and Jiří Kosinka. “Locally refinable gradient meshes supporting branching and sharp colour transitions”. In: *The Visual Computer* 34.6-8 (2018), pp. 949–960 (cit. on p. 156).
- [BSK19] Pieter J Barendrecht, Malcolm A Sabin and Jiří Kosinka. “A bivariate  $C^1$  subdivision scheme based on cubic half-box splines”. In: *Computer Aided Geometric Design* 71 (2019), pp. 77–89 (cit. on pp. 44, 63, 82).
- [BC16a] Michael Bartoň and Victor Manuel Calo. “Gaussian quadrature for splines via homotopy continuation: rules for  $C^2$  cubic splines”. In: *Journal of Computational and Applied Mathematics* 296 (2016), pp. 709–723 (cit. on p. 109).
- [BC16b] Michael Bartoň and Victor Manuel Calo. “Optimal quadrature rules for odd-degree spline spaces and their application to tensor-product-based isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 305 (2016), pp. 217–240 (cit. on p. 109).
- [BC17] Michael Bartoň and Victor Manuel Calo. “Gauss-Galerkin quadrature rules for quadratic and cubic spline spaces and their application to isogeometric analysis”. In: *Computer-Aided Design* 82 (2017), pp. 57–67 (cit. on p. 109).
- [Bat96] Klaus-Jürgen Bathe. *Finite element procedures*. Prentice Hall, 1996 (cit. on p. 88).
- [Bau72] Bruce G Baumgart. *Winged edge polyhedron representation*. Tech. rep. Stanford University, 1972 (cit. on p. 194).
- [Bec92] Adib Abdulghafour Becker. *The boundary element method in engineering: a complete course*. Vol. 19. McGraw-Hill London, 1992 (cit. on pp. 88, 127, 128, 131).
- [BCV02] Koen Beets, Johan Claes and Frank Van Reeth. “Borders, semi-sharp edges and adaptivity for hexagonal subdivision surface schemes”. In: *Advances in Modelling, Animation and Rendering*. Springer, 2002, pp. 151–166 (cit. on pp. 63, 72, 73).
- [BM17] Michel Bercovier and Tanya Matskewich. *Smooth Bézier Surfaces Over Unstructured Meshes*. Springer, 2017 (cit. on pp. 46, 56).
- [Bez+10] Hedlena Bezerra, Elmar Eisemann, Doug DeCarlo and Joëlle Thollot. “Diffusion constraints for vector graphics”. In: *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*. ACM, 2010, pp. 35–42 (cit. on p. 180).
- [BLZ00] Henning Biermann, Adi Levin and Denis Zorin. “Piecewise smooth subdivision surfaces with normal control”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 113–120 (cit. on p. 78).
- [Boc19] Sergey Anatolyevich Bochkhanov. *ALGLIB*. www.alglib.net. 2019 (cit. on p. 120).
- [Böh83] Wolfgang Böhm. “Generating the Bézier points of triangular splines”. In: *Surfaces in Computer Aided Geometric Design*. North-Holland Amsterdam, 1983, pp. 77–91 (cit. on p. 31).

- [BP93] Wolfgang Böhm and Hartmut Prautzsch. “Numerical Methods”. In: *AK Peters* (1993) (cit. on p. 58).
- [BZ04] Ioana Boier-Martin and Denis Zorin. “Differentiable parameterization of Catmull-Clark subdivision surfaces”. In: *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM. 2004, pp. 155–164 (cit. on pp. 106, 116).
- [Boo72] Carl de Boor. “On calculating with B-splines”. In: *Journal of Approximation Theory* 6.1 (1972), pp. 50–62 (cit. on p. 16).
- [BHR93] Carl de Boor, Klaus Höllig and Sherman Riemenschneider. *Box splines*. Vol. 98. Springer, 1993 (cit. on pp. 27, 29).
- [Bor+11] Michael J Borden, Michael A Scott, John A Evans and Thomas JR Hughes. “Isogeometric finite element data structures based on Bézier extraction of NURBS”. In: *International Journal for Numerical Methods in Engineering* 87.1-5 (2011), pp. 15–47 (cit. on p. 101).
- [BGH03] Steffen Börm, Lars Grasedyck and Wolfgang Hackbusch. “Introduction to hierarchical matrices with applications”. In: *Engineering analysis with boundary elements* 27.5 (2003), pp. 405–422 (cit. on p. 132).
- [Bos+12] Maria Boschioli, Christoph Fünfzig, Lucia Romani and Gudrun Albrecht. “ $G^1$  rational blend interpolatory schemes: a comparative study”. In: *Graphical models* 74.1 (2012), pp. 29–49 (cit. on p. 58).
- [BA08] Tamy Boubekeur and Marc Alexa. “Phong Tessellation”. In: *ACM Transactions on Graphics* 27.5 (2008), 141:1–141:5 (cit. on p. 147).
- [BLW11] John C Bowers, Jonathan Leahey and Rui Wang. “A ray tracing approach to diffusion curves”. In: *Computer Graphics Forum*. Vol. 30. 4. Wiley Online Library. 2011, pp. 1345–1352 (cit. on p. 180).
- [BGR10] James Bremer, Zydrunas Gimbutas and Vladimir Rokhlin. “A nonlinear optimization procedure for generalized Gaussian quadratures”. In: *SIAM Journal on Scientific Computing* 32.4 (2010), pp. 1761–1788 (cit. on p. 118).
- [BS08] Susanne C Brenner and Larkin Ridgway Scott. *The mathematical theory of finite element methods*. Springer, 2008 (cit. on p. 99).
- [Bur11] Daniel Burkhart. “Subdivision for Volumetric Finite Elements”. PhD thesis. University of Kaiserslautern, 2011 (cit. on p. 103).
- [BHU10] Daniel Burkhart, Bernd Hamann and Georg Umlauf. “Isogeometric Finite Element Analysis Based on Catmull-Clark Subdivision Solids”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1575–1584 (cit. on p. 103).
- [BF01] Samuel R Buss and Jay P Fillmore. “Spherical averages and applications to spherical splines and interpolation”. In: *ACM Transactions on Graphics* 20.2 (2001), pp. 95–126 (cit. on p. 151).
- [CST17] Francesco Calabrò, Giancarlo Sangalli and Mattia Tani. “Fast formation of isogeometric Galerkin matrices by weighted quadrature”. In: *Computer Methods in Applied Mechanics and Engineering* 316 (2017), pp. 606–622 (cit. on p. 109).



- [Cam17] Marcel Campen. “Partitioning surfaces into quadrilateral patches: a survey”. In: *Computer Graphics Forum*. Vol. 36. 8. Wiley Online Library, 2017, pp. 567–588 (cit. on p. 117).
- [CBK12] Marcel Campen, David Bommers and Leif Kobbelt. “Dual loops meshing: quality quad layouts on manifolds”. In: *ACM Transactions on Graphics* 31.4 (2012), p. 110 (cit. on p. 117).
- [Cas10] Thomas J Cashman. “NURBS-compatible subdivision surfaces”. PhD thesis. University of Cambridge, 2010 (cit. on p. 83).
- [Cas12] Thomas J Cashman. “Beyond Catmull-Clark? A survey of advances in subdivision surface methods”. In: *Computer Graphics Forum* 31 (2012), pp. 42–61 (cit. on p. 61).
- [CC78] Edwin Catmull and James Clark. “Recursively generated B-spline surfaces on arbitrary topological meshes”. In: *Computer-Aided Design* 10.6 (1978), pp. 350–355 (cit. on pp. 61, 63, 78, 82).
- [CB10] Dominique Chapelle and Klaus-Jürgen Bathe. *The finite element analysis of shells-fundamentals*. Springer Science & Business Media, 2010 (cit. on p. 88).
- [Chi86] Hiroaki Chiyokura. “Localized surface interpolation method for irregular meshes”. In: *Advanced Computer Graphics*. Springer, 1986, pp. 3–19 (cit. on p. 61).
- [CK83] Hiroaki Chiyokura and Fumihiko Kimura. “Design of solids with free-form surfaces”. In: *ACM SIGGRAPH Computer Graphics* 17.3 (1983), pp. 289–298 (cit. on pp. 46, 58, 59, 61).
- [CO01] Fehmi Cirak and Michael Ortiz. “Fully  $C^1$ -conforming subdivision elements for finite deformation thin-shell analysis”. In: *International Journal for Numerical Methods in Engineering* 51.7 (2001), pp. 813–833 (cit. on p. 103).
- [COS00] Fehmi Cirak, Michael Ortiz and Peter Schröder. “Subdivision surfaces: a new paradigm for thin-shell finite-element analysis”. In: *International Journal for Numerical Methods in Engineering* 47.12 (2000), pp. 2039–2072 (cit. on p. 103).
- [Cir+02] Fehmi Cirak, Michael J Scott, Erik K Antonsson, Michael Ortiz and Peter Schröder. “Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision”. In: *Computer-Aided Design* 34.2 (2002), pp. 137–148 (cit. on p. 103).
- [CBV02] Johan Claes, Koen Beets and Frank Van Reeth. “A corner-cutting scheme for hexagonal subdivision surfaces”. In: *Shape Modeling International, 2002. Proceedings*. IEEE, 2002, pp. 13–20 (cit. on p. 63).
- [CHB09] J Austin Cottrell, Thomas JR Hughes and Yuri Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. John Wiley & Sons, 2009 (cit. on pp. 88, 100, 103, 132).
- [Cox72] Maurice G Cox. “The numerical evaluation of B-splines”. In: *IMA Journal of Applied Mathematics* 10.2 (1972), pp. 134–149 (cit. on p. 16).

- [DM83] Wolfgang Dahmen and Charles A Micchelli. “Recent progress in multivariate splines”. In: *Approximation Theory IV*. Ed. by Charles K Chui, Larry L Schumaker and Joseph D Ward. New York, Academic Press, 1983, pp. 27–121 (cit. on p. 26).
- [DMS92] Wolfgang Dahmen, Charles A Micchelli and Hans-Peter Seidel. “Blossoming begets B-spline bases built better by B-patches”. In: *Mathematics of computation* 59.199 (1992), pp. 97–115 (cit. on p. 28).
- [Dav11] Alan J Davies. *The finite element method: An introduction with partial differential equations*. Oxford University Press, 2011 (cit. on pp. 130, 131).
- [Deg90] Wendelin LF Degen. “Explicit continuity conditions for adjacent Bézier surface patches”. In: *Computer Aided Geometric Design* 7.1-4 (1990), pp. 181–189 (cit. on p. 46).
- [DeR90] Tony D DeRose. “Necessary and sufficient conditions for tangent plane continuity of Bézier surfaces”. In: *Computer Aided Geometric Design* 7.1-4 (1990), pp. 165–179 (cit. on pp. 48, 58).
- [DSO12] Engin Dikici, Sten Roar Snare and Fredrik Orderud. “Isoparametric finite element analysis for Doo-Sabin subdivision models”. In: *Proceedings of the 2012 Graphics Interface Conference*. Canadian Information Processing Society. 2012, pp. 19–26 (cit. on p. 103).
- [Dod05] Neil A Dodgson. “An heuristic analysis of the classification of bivariate subdivision schemes”. In: *Mathematics of Surfaces XI*. Springer, 2005, pp. 161–183 (cit. on p. 64).
- [Dod+09] Neil A Dodgson, Ursula H Augsdörfer, Thomas J Cashman and Malcolm A Sabin. “Deriving box-spline subdivision schemes”. In: *Mathematics of Surfaces XIII*. Springer. 2009, pp. 106–123 (cit. on pp. 61, 81).
- [Don+16] Marco Donatelli, Paola Novara, Lucia Romani, Stefano Serra-Capizzano and Debora Sesana. *Surface Subdivision Algorithms and Structured Linear Algebra: a Computational Approach to Determine Bounds of Extraordinary Rule Weights*. Tech. rep. Tech. Rep. 2016-012, Department of Information Technology, Uppsala University, 2016 (cit. on p. 69).
- [DS78] Daniel WH Doo and Malcolm A Sabin. “Behaviour of recursive division surfaces near extraordinary points”. In: *Computer-Aided Design* 10.6 (1978), pp. 356–360 (cit. on pp. 61, 66, 70, 78, 82).
- [DFG99] Qiang Du, Vance Faber and Max Gunzburger. “Centroidal Voronoi tessellations: Applications and algorithms”. In: *SIAM Review* 41.4 (1999), pp. 637–676 (cit. on p. 193).
- [Du88] Wen-Hui Du. “Etude sur la représentation des surfaces complexes: application à la reconstruction de surfaces échantillonnées”. PhD thesis. Télécom Paris, 1988 (cit. on p. 46).
- [DS88] Wen-Hui Du and Francis JM Schmitt. “New results for the smooth connection between tensor product Bezier patches”. In: *New Trends in Computer Graphics*. Springer, 1988, pp. 351–363 (cit. on pp. 48, 51, 52).

- [DS90] Wen-Hui Du and Francis JM Schmitt. “On the  $G^1$  continuity of piecewise Bézier surfaces: a review with new results”. In: *Computer-Aided Design* 22.9 (1990), pp. 556–573 (cit. on p. 48).
- [DS91] Wen-Hui Du and Francis JM Schmitt. “ $G^1$  Smooth Connection Between Rectangular and Triangular Bézier Patches at a Common Corner”. In: *Curves and surfaces*. Elsevier, 1991, pp. 165–168 (cit. on pp. 48, 53).
- [Dun85] David A Dunavant. “High degree efficient symmetrical Gaussian quadrature rules for the triangle”. In: *International journal for numerical methods in engineering* 21.6 (1985), pp. 1129–1148 (cit. on p. 96).
- [DLG87] Nira Dyn, David Levin and John A Gregory. “A 4-point interpolatory subdivision scheme for curve design”. In: *Computer aided geometric design* 4.4 (1987), pp. 257–268 (cit. on p. 173).
- [DLG90] Nira Dyn, David Levin and John A Gregory. “A butterfly subdivision scheme for surface interpolation with tension control”. In: *ACM transactions on Graphics* 9.2 (1990), pp. 160–169 (cit. on p. 61).
- [DLL92] Nira Dyn, David Levin and Dingyuan Liu. “Interpolatory convexity-preserving subdivision schemes for curves and surfaces”. In: *Computer-Aided Design* 24.4 (1992), pp. 211–216 (cit. on p. 64).
- [DLS03] Nira Dyn, David Levin and Jo Simoens. “Face value subdivision schemes on triangulations by repeated averaging”. In: *Curve and Surface Fitting: Saint Malo*. 2003, pp. 129–138 (cit. on p. 64).
- [EDH10] Anand Embar, John Dolbow and Isaac Harari. “Imposing Dirichlet boundary conditions with Nitsche’s method and spline-based finite elements”. In: *International journal for numerical methods in engineering* 83.7 (2010), pp. 877–898 (cit. on p. 101).
- [Epp+08] David Eppstein, Michael T Goodrich, Ethan Kim and Rasmus Tamstorf. “Motorcycle graphs: canonical quad mesh partitioning”. In: *Computer Graphics Forum*. Vol. 27. 5. Wiley Online Library. 2008, pp. 1477–1486 (cit. on p. 117).
- [Ern11] Dennis Ernens. “Finite element methods with exact geometry representation”. MA thesis. Delft University of Technology, 2011 (cit. on p. 134).
- [Far82a] Gerald E Farin. “A construction for visual  $C^1$  continuity of polynomial surface patches”. In: *Computer Graphics and Image Processing* 19.1 (1982), p. 94 (cit. on p. 46).
- [Far82b] Gerald E Farin. “Designing  $C^1$  surfaces consisting of triangular cubic patches”. In: *Computer-Aided Design* 14.5 (1982), pp. 253–256 (cit. on p. 31).
- [Far83] Gerald E Farin. “Smooth interpolation to scattered 3D-data”. In: *Surfaces in CAGD*. Ed. by Robert. E. Barnhill and Wolfgang Böhm. 1983 (cit. on p. 58).
- [Far91] Gerald E Farin. *NURBS for curve and surface design*. Society for Industrial and Applied Mathematics, 1991 (cit. on p. 16).
- [Far02] Gerald E Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002 (cit. on pp. 10, 159, 160, 161).

- [FH12] Gerald E Farin and Dianne Hansford. “Agnostic  $G^1$  Gregory surfaces”. In: *Graphical Models* 74.6 (2012), pp. 346–350 (cit. on p. 61).
- [FHK02] Gerald E Farin, Josef Hoschek and Myung-Soo Kim. *Handbook of Computer Aided Geometric Design*. Amsterdam: Elsevier, 2002 (cit. on p. 10).
- [FSH11] Mark Finch, John Snyder and Hugues Hoppe. “Freeform vector graphics with controlled thin-plate splines”. In: *ACM Transactions on Graphics*. Vol. 30. 6. ACM. 2011, p. 166 (cit. on p. 180).
- [FB07] Jacob Fish and Ted Belytschko. *A first course in finite elements*. Wiley, 2007 (cit. on p. 97).
- [Flo03] Michael S Floater. “Mean value coordinates”. In: *Computer aided geometric design* 20.1 (2003), pp. 19–27 (cit. on p. 145).
- [Flo15] Michael S Floater. “Generalized barycentric coordinates and applications”. In: *Acta Numerica* 24 (2015), pp. 161–214 (cit. on pp. 145, 151).
- [FB88] David R Forsey and Richard H Bartels. “Hierarchical B-spline refinement”. In: *ACM Siggraph Computer Graphics* 22.4 (1988), pp. 205–212 (cit. on pp. 125, 190).
- [Fre71] Paul O Frederickson. *Generalized Triangular Splines, Mathematics Report 7-71*. Tech. rep. Lakehead University, 1971 (cit. on p. 31).
- [GKW13] Lothar Gaul, Martin Kögl and Marcus Wagner. *Boundary element methods for engineers and scientists: an introductory course with advanced topics*. Springer Science & Business Media, 2013 (cit. on pp. 128, 131).
- [GJS12] Carlotta Giannelli, Bert Jüttler and Hendrik Speleers. “THB-splines: The truncated basis for hierarchical splines”. In: *Computer Aided Geometric Design* 29.7 (2012), pp. 485–498 (cit. on pp. 125, 192).
- [Gin+14] Alexandros I Ginnis, Konstantinos V Kostas, Constantinos G Politis, Panagiotis D Kaklis, Kostas A Belibassakis, Theodoros P Gerostathis, Michael A Scott and Thomas JR Hughes. “Isogeometric boundary-element analysis for the wave-resistance problem using T-splines”. In: *Computer Methods in Applied Mechanics and Engineering* 279 (2014), pp. 425–439 (cit. on p. 132).
- [Gol02] Ron Goldman. *Pyramid algorithms: A dynamic programming approach to curves and surfaces for geometric modeling*. Morgan Kaufmann, 2002 (cit. on p. 17).
- [Gre+17] Francesco Greco, Pieter J Barendrecht, Laurens Coox, Onur Atak and Wim Desmet. “Finite element analysis enhanced with subdivision surface boundary representations”. In: *Finite Elements in Analysis and Design* 137 (2017), pp. 56–72 (cit. on pp. 86, 132).
- [Gre74] John A Gregory. “Smooth interpolation without twist constraints”. In: *Computer aided geometric design*. Elsevier, 1974, pp. 71–87 (cit. on pp. 46, 59).
- [HB00] Stefanie Hahmann and Georges-Pierre Bonneau. “Triangular  $G^1$  interpolation by 4-splitting domain triangles”. In: *Computer Aided Geometric Design* 17.8 (2000), pp. 731–757 (cit. on p. 58).

- [HBC08] Stefanie Hahmann, Georges-Pierre Bonneau and Baptiste Caramiaux. “Bicubic  $G^1$  interpolation of irregular quad meshes using a 4-split”. In: *International Conference on Geometric Modeling and Processing*. Springer. 2008, pp. 17–32 (cit. on p. 58).
- [HR16] Jan Hakenberg and Ulrich Reif. “On the volume of sets bounded by refinable functions”. In: *Applied Mathematics and Computation* 272 (2016), pp. 2–19 (cit. on p. 113).
- [HKD93] Mark Halstead, Michael Kass and Tony DeRose. “Efficient, fair interpolation using Catmull-Clark surfaces”. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM. 1993, pp. 35–44 (cit. on p. 104).
- [HBK18] Gerben J Hettinga, Pieter J Barendrecht and Jiri Kosinka. “A Comparison of GPU Tessellation Strategies for Multisided Patches.” In: *Eurographics (Short Papers)*. 2018, pp. 45–48 (cit. on p. 138).
- [HK17] Gerben J Hettinga and Jiri Kosinka. “Phong Tessellation and PN Polygons for Polygonal Models.” In: *Eurographics (Short Papers)*. 2017, pp. 49–52 (cit. on p. 147).
- [HK18] Gerben J Hettinga and Jiri Kosinka. “Multisided generalisations of Gregory patches”. In: *Computer Aided Geometric Design* 62 (2018), pp. 166–180 (cit. on p. 149).
- [Höl03] Klaus Höllig. *Finite element methods with B-splines*. Vol. 26. Siam, 2003 (cit. on p. 103).
- [HL93] Josef Hoschek and Dieter Lasser. *Fundamentals of computer aided geometric design*. AK Peters, Ltd., 1993 (cit. on p. 10).
- [Hub+17] Simeon Hubrich, Paolo Di Stolfo, László Kudela, Stefan Kollmannsberger, Ernst Rank, Andreas Schröder and Alexander Düster. “Numerical integration of discontinuous functions: moment fitting and smart octree”. In: *Computational Mechanics* 60.5 (2017), pp. 863–881 (cit. on p. 118).
- [Hug87] Thomas JR Hughes. “The finite element method: linear static and dynamic finite element analysis”. In: *Prentice-Hall* (1987) (cit. on p. 88).
- [HCB05] Thomas JR Hughes, John A Cottrell and Yuri Bazilevs. “Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement”. In: *Computer methods in applied mechanics and engineering* 194.39 (2005), pp. 4135–4195 (cit. on pp. 101, 103).
- [HRS10] Thomas JR Hughes, Alessandro Reali and Giancarlo Sangalli. “Efficient quadrature for NURBS-based isogeometric analysis”. In: *Computer methods in applied mechanics and engineering* 199.5 (2010), pp. 301–313 (cit. on pp. 109, 116).
- [Ilb+13] Peter Ilbery, Luke Kendall, Cyril Concolato and Michael McCosker. “Biharmonic diffusion curve images from boundary elements”. In: *ACM Transactions on Graphics* 32.6 (2013), p. 219 (cit. on p. 179).

- [IR72] Bruce M Irons and Abdur Razzaque. “Experience with the patch test for convergence of finite elements”. In: *The mathematical foundations of the finite element method with applications to partial differential equations*. Elsevier, 1972, pp. 557–587 (cit. on p. 97).
- [Jes16] Stefan Jeschke. “Generalized Diffusion Curves: An Improved Vector Representation for Smooth-Shaded Images”. In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library, 2016, pp. 71–79 (cit. on p. 180).
- [Joh12] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012 (cit. on pp. 88, 99).
- [Joh19] Steven G. Johnson. *The NLOpt nonlinear-optimization package*. <http://github.com/stevengj/nlop>. 2019 (cit. on p. 120).
- [Jos+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green and Tom Sanocki. “Harmonic coordinates for character articulation”. In: *ACM Transactions on Graphics* 26.3 (2007), p. 71 (cit. on p. 145).
- [Jüt+16] Bert Jüttler, Angelos Mantzaflaris, Ricardo Perl and Martin Rumpf. “On numerical integration in isogeometric subdivision methods for PDEs on surfaces”. In: *Computer Methods in Applied Mechanics and Engineering* 302 (2016), pp. 131–146 (cit. on p. 109).
- [Kan+17] Tadej Kanduč, Carlotta Giannelli, Francesca Pelosi and Hendrik Speleers. “Adaptive isogeometric analysis with hierarchical box splines”. In: *Computer Methods in Applied Mechanics and Engineering* 316 (2017), pp. 817–838 (cit. on p. 191).
- [KP07] Kęstutis Karčiauskas and Jörg Peters. “Concentric tessellation maps and curvature continuous guided surfaces”. In: *Computer Aided Geometric Design* 24.2 (2007), pp. 99–111 (cit. on p. 83).
- [KP09] Kęstutis Karčiauskas and Jörg Peters. “Adjustable speed surface subdivision”. In: *Computer Aided Geometric Design* 26.9 (2009), pp. 962–969 (cit. on p. 116).
- [KP18] Kęstutis Karčiauskas and Jörg Peters. “Rapidly contracting subdivision yields finite, effectively C2 surfaces”. In: *Computers & Graphics* 74 (2018), pp. 182–190 (cit. on p. 116).
- [Kat02] John T Katsikadelis. *Boundary elements: theory and applications*. Elsevier, 2002 (cit. on p. 127).
- [KSS16] John Kessenich, Graham Sellers and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. Addison-Wesley Professional, 2016 (cit. on p. 140).
- [Kic16] Przemysław Kiciak. *Geometric Continuity of Curves and Surfaces*. Vol. 8. 3. Morgan & Claypool Publishers, 2016, pp. 1–249 (cit. on p. 46).
- [Kob00] Leif Kobbelt. “ $\sqrt{3}$ -subdivision”. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 103–112 (cit. on pp. 61, 193).
- [KB09] Tamara G Kolda and Brett W Bader. “Tensor decompositions and applications”. In: *SIAM review* 51.3 (2009), pp. 455–500 (cit. on p. 196).

- [KSD13] Jiří Kosinka, Malcolm A Sabin and Neil Dodgson. “Cubic subdivision schemes with double knots”. In: *Computer Aided Geometric Design* 30.1 (2013), pp. 45–57 (cit. on pp. 72, 73).
- [KSD14] Jiří Kosinka, Malcolm A Sabin and Neil Dodgson. “Creases and boundary conditions for subdivision curves”. In: *Graphical Models* 76.5 (2014), pp. 240–251 (cit. on p. 116).
- [KSD15] Jiří Kosinka, Malcolm A Sabin and Neil A Dodgson. “Control vectors for splines”. In: *Computer-Aided Design* 58 (2015), pp. 173–178 (cit. on p. 173).
- [Kra97] Rainer Kraft. “Adaptive and linearly independent multilevel B-splines”. In: *Surface Fitting and Multiresolution Methods*. Vanderbilt University Press, Nashville, 1997, pp. 209–218 (cit. on p. 190).
- [Kru00] Pavel A Krutitskii. “The Dirichlet problem for the two-dimensional Laplace equation in a multiply connected domain with cuts”. In: *Proceedings of the Edinburgh Mathematical Society* 43.2 (2000), pp. 325–341 (cit. on p. 179).
- [Kur+13] Göktürk Kuru, Clemens V Verhoosel, Kristoffer G van der Zee and E Harald van Brummelen. “Goal-adaptive Isogeometric Analysis with hierarchical splines”. MA thesis. Eindhoven University of Technology, 2013 (cit. on p. 191).
- [LB07] Dylan Lacewell and Brent Burley. “Exact Evaluation of Catmull-Clark Subdivision Surfaces Near B-Spline Boundaries”. In: *Journal of Graphics, GPU, and Game Tools* 12.3 (2007), pp. 7–15 (cit. on pp. 75, 116).
- [LS07] Ming-Jun Lai and Larry L Schumaker. *Spline functions on triangulations*. Vol. 110. Cambridge University Press, 2007 (cit. on p. 83).
- [LBS06] Torsten Langer, Alexander Belyaev and Hans-Peter Seidel. “Spherical barycentric coordinates”. In: *Symposium on Geometry Processing*. 2006, pp. 81–88 (cit. on p. 151).
- [LB93] Nam-Sua Lee and Klaus-Jürgen Bathe. “Effects of element distortions on the performance of isoparametric elements”. In: *International Journal for numerical Methods in engineering* 36.20 (1993), pp. 3553–3576 (cit. on pp. 92, 97).
- [Lev09] Raphael L Levien. “From Spiral to Spline: Optimal Techniques in Interactive Curve Design”. PhD thesis. EECS Department, University of California, Berkeley, Dec. 2009 (cit. on p. 13).
- [LS09] Raphael L Levien and Carlo H Séquin. “Interpolating splines: which is the fairest of them all?” In: *Computer-Aided Design and Applications* 6.1 (2009), pp. 91–102 (cit. on p. 160).
- [Li+12] Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang and Baining Guo. “All-hex meshing using singularity-restricted field”. In: *ACM Transactions on Graphics* 31.6 (2012), p. 177 (cit. on p. 126).
- [Lie+17] Henrik Lieng, Jiří Kosinka, Jingjing Shen and Neil A Dodgson. “A Colour Interpolation Scheme for Topologically Unrestricted Gradient Meshes”. In: *Computer Graphics Forum*. Vol. 36. 6. Wiley Online Library. 2017, pp. 112–121 (cit. on p. 173).



- [Liu86] Ding-yuan Liu. “A geometric condition for smoothness between adjacent Bézier surface patches”. In: *Acta Math. Appl. Sinica* 9.4 (1986), pp. 432–442 (cit. on p. 48).
- [LH89] Ding-yuan Liu and Josef Hoschek. “GC1 continuity conditions between adjacent rectangular and triangular Bézier surface patches”. In: *Computer-Aided Design* 21.4 (1989), pp. 194–200 (cit. on pp. 46, 58).
- [Liu+14] Lei Liu, Yongjie Zhang, Thomas JR Hughes, Michael A Scott and Thomas W Sederberg. “Volumetric T-spline construction using Boolean operations”. In: *Engineering with Computers* 30.4 (2014), pp. 425–439 (cit. on p. 126).
- [Liu+09] Yang Liu, Wenping Wang, Bruno Lévy, Feng Sun, Dong-Ming Yan, Lin Lu and Chenglei Yang. “On centroidal voronoi tessellation—energy smoothness and fast computation”. In: *ACM Transactions on Graphics* 28.4 (2009), p. 101 (cit. on p. 194).
- [Liu09] Yijun Liu. *Fast multipole boundary element method: theory and applications in engineering*. Cambridge university press, 2009 (cit. on p. 132).
- [Llo82] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137 (cit. on p. 193).
- [Lon85] Lucia Longhi. *Interpolating patches between cubic boundaries*. Tech. rep. University of California at Berkeley, 1985 (cit. on pp. 59, 84).
- [Loo87] Charles T Loop. “Smooth subdivision surfaces based on triangles”. MA thesis. University of Utah, 1987 (cit. on pp. 61, 82).
- [Loo94] Charles T Loop. “Smooth spline surfaces over irregular meshes”. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 303–310 (cit. on pp. 46, 53, 57).
- [LD89] Charles T Loop and Tony D DeRose. “A multisided generalization of Bézier surfaces”. In: *ACM Transactions on Graphics* 8.3 (1989), pp. 204–234 (cit. on pp. 25, 147).
- [LS08] Charles T Loop and Scott Schaefer. “Approximating Catmull-Clark subdivision surfaces with bicubic patches”. In: *ACM Transactions on Graphics* 27.1 (2008), p. 8 (cit. on p. 83).
- [Loo+09] Charles T Loop, Scott Schaefer, Tianyun Ni and Ignacio Castaño. “Approximating subdivision surfaces with Gregory patches for hardware tessellation”. In: *ACM Transactions on Graphics*. Vol. 28. 5. ACM. 2009, p. 151 (cit. on pp. 83, 84).
- [LJ75] James N Lyness and Dennis C Jespersen. “Moderate degree symmetric quadrature rules for the triangle”. In: *IMA Journal of Applied Mathematics* 15.1 (1975), pp. 19–32 (cit. on p. 118).
- [Ma05] Weiyin Ma. “Subdivision surfaces for CAD – an overview”. In: *Computer-Aided Design* 37.7 (2005), pp. 693–709 (cit. on p. 61).
- [Man06] Stephen Mann. “A blossoming development of splines”. In: *Synthesis Lectures on Computer Graphics and Animation* 1.1 (2006), pp. 1–108 (cit. on p. 17).



- [Man+92] Stephen Mann, Charles T Loop, Michael Lounsbery, David Meyers, James Painter, Tony Derosé and Kenneth Sloan. “A survey of parametric scattered data fitting using triangular interpolants”. In: *Curve and Surface Design*. SIAM, 1992, pp. 145–172 (cit. on p. 58).
- [Män88] Martti Mäntylä. *An introduction to solid modeling*. Computer science press, 1988 (cit. on p. 194).
- [MJ15] Angelos Mantzaflaris and Bert Jüttler. “Integration by interpolation and look-up for Galerkin-based isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 284 (2015), pp. 373–400 (cit. on p. 109).
- [ML95] Ueda Masami and Suresh Lodha. *Wavelets: An elementary introduction and examples*. Tech. rep. University of California at Santa Cruz, 1995 (cit. on p. 41).
- [MP77] Charles A Micchelli and Allan Pinkus. “Moment theory for weak Chebyshev systems with applications to monosplines, quadrature formulae and best one-sided  $L^1$  approximation by spline functions with fixed knots”. In: *SIAM J. Math. Anal.* 8 (1977), pp. 206–230 (cit. on p. 109).
- [MGT11] Toby J Mitchell, Sanjay Govindjee and Robert L Taylor. “A method for enforcement of Dirichlet boundary conditions in isogeometric analysis”. In: *Recent Developments and Innovative Applications in Computational Mechanics*. Springer, 2011, pp. 283–293 (cit. on p. 101).
- [Mun+11] Aaftab Munshi, Benedict Gaster, Timothy G Mattson and Dan Ginsburg. *OpenCL programming guide*. Pearson Education, 2011 (cit. on p. 141).
- [MP09] Ashish Myles and Jörg Peters. “Bi-3  $C^2$  polar subdivision”. In: *ACM Transactions on Graphics*. Vol. 28. 3. ACM. 2009, p. 48 (cit. on p. 83).
- [NKP14] Thien Nguyen, Keçstutis Karčiauskas and Jörg Peters. “A comparative study of several classical, discrete differential and isogeometric methods for solving Poisson’s equation on the disk”. In: *Axioms* 3.2 (2014), pp. 280–299 (cit. on p. 109).
- [Ops13] Timo M van Opstal. “Numerical methods for inflatables with multiscale geometries”. PhD thesis. Eindhoven University of Technology, 2013 (cit. on p. 132).
- [OBZ15] Timo M van Opstal, E Harald van Brummelen and Gertjan van Zwieten. “A finite-element/boundary-element method for three-dimensional, large-displacement fluid–structure-interaction”. In: *Computer Methods in Applied Mechanics and Engineering* 284 (2015), pp. 637–663 (cit. on p. 132).
- [Orz+08] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot and David Salesin. “Diffusion Curves: A Vector Representation for Smooth-shaded Images”. In: *ACM Transactions on Graphics* 27.3 (2008), 92:1–92:8 (cit. on pp. 158, 177, 178).
- [OS03] Peter Oswald and Peter Schröder. “Composite primal/dual  $\sqrt{3}$  subdivision schemes”. In: *Computer Aided Geometric Design* 20.3 (2003), pp. 135–164 (cit. on p. 64).

- [Pan+12] Wai-Man Pang, Jing Qin, Michael Cohen, Pheng-Ann Heng and Kup-Sze Choi. “Fast rendering of diffusion curves with triangles”. In: *IEEE Computer Graphics and Applications* 32.4 (2012), pp. 68–78 (cit. on p. 180).
- [PB+12] Paul William Partridge, Carlos Alberto Brebbia et al. *Dual reciprocity boundary element method*. Springer Science & Business Media, 2012 (cit. on p. 128).
- [PS04a] Per-Olof Persson and Gilbert Strang. “A Simple Mesh Generator in MATLAB”. In: *SIAM Review* 46.2 (2004), pp. 329–345 (cit. on p. 135).
- [Pet90a] Jörg Peters. “Local smooth surfaces interpolation: a classification”. In: *Computer Aided Geometric Design* 7 (1990), pp. 191–195 (cit. on pp. 46, 56, 58).
- [Pet90b] Jörg Peters. “Smooth mesh interpolation with cubic patches”. In: *Computer-Aided Design* 22.2 (1990), pp. 109–120 (cit. on p. 58).
- [Pet91a] Jörg Peters. “Parametrizing singularly to enclose data points by a smooth parametric surface”. In: *Proceedings of Graphics Interface*. Vol. 91. 1991 (cit. on p. 58).
- [Pet91b] Jörg Peters. “Smooth interpolation of a mesh of curves”. In: *Constructive Approximation* 7.1 (1991), pp. 221–246 (cit. on pp. 46, 51, 55).
- [Pet02] Jörg Peters. “Geometric continuity”. In: *Handbook of Computer Aided Geometric Design*. Ed. by Gerald E Farin, Josef Hoschek and Myung-Soo Kim. Elsevier, 2002. Chap. 8, pp. 193–229 (cit. on pp. 46, 58).
- [Pet14] Jörg Peters. “Refinability of splines derived from regular tessellations”. In: *Computer Aided Geometric Design* 31.3-4 (2014), pp. 141–147 (cit. on p. 83).
- [PF10] Jörg Peters and Jianhua Fan. “On the complexity of smooth spline surfaces from quad meshes”. In: *Computer Aided Geometric Design* 27.1 (2010), pp. 96–105 (cit. on p. 56).
- [PR97] Jörg Peters and Ulrich Reif. “The simplest subdivision scheme for smoothing polyhedra”. In: *ACM Transactions on Graphics* 16.4 (1997), pp. 420–431 (cit. on pp. 61, 82).
- [PR98] Jörg Peters and Ulrich Reif. “Analysis of algorithms generalizing B-spline subdivision”. In: *SIAM Journal on Numerical Analysis* 35.2 (1998), pp. 728–748 (cit. on p. 78).
- [PR08] Jörg Peters and Ulrich Reif. *Subdivision surfaces*. Springer, 2008 (cit. on pp. 61, 66, 69, 76, 77).
- [PS04b] Jörg Peters and Le-Jeng Shiue. “Combining 4-and 3-direction subdivision”. In: *ACM Transactions on Graphics* 23.4 (2004), pp. 980–1003 (cit. on p. 82).
- [PW06] Jörg Peters and Xiaobin Wu. “On the local linear independence of generalized subdivision functions”. In: *SIAM Journal on Numerical Analysis* 44.6 (2006), pp. 2389–2407 (cit. on p. 125).
- [PT97] Les A Piegl and Wayne Tiller. *The NURBS book*. Springer, 1997 (cit. on p. 16).
- [Pip87] Bruce R Piper. “Visually smooth interpolation with triangular Bézier patches”. In: *Geometric modeling: algorithms and new trends* (1987), pp. 221–233 (cit. on pp. 46, 58).

- [Poz02] Constantine Pozrikidis. *A practical guide to boundary element methods with the software library BEMLIB*. CRC Press, 2002 (cit. on pp. 127, 128, 130).
- [PB02] Harmut Prautzsch and Wolfgang Böhm. “Box splines”. In: *Handbook of Computer Aided Geometric Design*. Ed. by Gerald E Farin, Josef Hoschek and Myung-Soo Kim. Elsevier, 2002. Chap. 10, pp. 255–283 (cit. on pp. 27, 63).
- [Pra84] Hartmut Prautzsch. “Unterteilungsalgorithmen für multivariate Splines – ein geometrischer Zugang”. PhD thesis. Technische Universität Braunschweig, 1984 (cit. on p. 31).
- [PBP02] Hartmut Prautzsch, Wolfgang Böhm and Marco Paluszny. *Bézier and B-spline techniques*. Springer, 2002 (cit. on pp. 10, 27).
- [PU99] Hartmut Prautzsch and Georg Umlauf. “Triangular  $G^2$  splines”. In: *Curves and Surface Design*. 1999, pp. 335–342 (cit. on p. 64).
- [PJS15] Romain Prévost, Wojciech Jarosz and Olga Sorkine-Hornung. “A vectorial framework for ray traced diffusion curves”. In: *Computer Graphics Forum*. Vol. 34. 1. Wiley Online Library. 2015, pp. 253–264 (cit. on pp. 179, 180).
- [Pro19] Christopher G Provatidis. *Precursors of Isogeometric Analysis*. Springer, 2019 (cit. on p. 103).
- [Ram89] Lyle Ramshaw. “Blossoms are polar forms”. In: *Computer Aided Geometric Design* 6.4 (1989), pp. 323–358 (cit. on pp. 17, 73).
- [Rei95] Ulrich Reif. “A unified approach to subdivision algorithms near extraordinary vertices”. In: *Computer Aided Geometric Design* 12.2 (1995), pp. 153–174 (cit. on p. 66).
- [RS01] Ulrich Reif and Peter Schröder. “Curvature integrability of subdivision surfaces”. In: *Advances in Computational Mathematics* 14.2 (2001), pp. 157–174 (cit. on p. 106).
- [Rog00] David F Rogers. *An introduction to NURBS: with historical perspective*. Elsevier, 2000 (cit. on p. 16).
- [Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003 (cit. on p. 96).
- [Sab77] Malcolm A Sabin. “The use of piecewise forms for the numerical representation of shape”. PhD thesis. Hungarian Academy of Science, Budapest, 1977 (cit. on p. 31).
- [Sab10] Malcolm A Sabin. *Analysis and design of univariate subdivision schemes*. Vol. 6. Springer, 2010 (cit. on pp. 61, 116).
- [SB03] Malcolm A Sabin and Aurelian Bejancu. “Boundary conditions for the 3-direction box-spline”. In: *Mathematics of Surfaces*. Springer, 2003, pp. 244–261 (cit. on p. 72).
- [Sab+07] Malcolm A Sabin, Thomas J Cashman, Ursula H Augsdörfer and Neil A Dodgson. “Bounded curvature subdivision without eigenanalysis”. In: *Mathematics of surfaces XII*. Springer, 2007, pp. 391–411 (cit. on p. 116).

- [Sal12] Lee Sallows. “On self-tiling tile sets”. In: *Mathematics Magazine* 85.5 (2012), pp. 323–333 (cit. on p. 83).
- [Sal14] Lee Sallows. “More On Self-Tiling Tile Sets”. In: *Mathematics Magazine* 87.2 (2014), pp. 100–112 (cit. on p. 83).
- [Sal06] David Salomon. *Curves and surfaces for computer graphics*. Springer New York, 2006 (cit. on p. 10).
- [SK10] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010 (cit. on p. 141).
- [Sar87] Ramon F Sarraga. “ $G^1$  interpolation of generally unrestricted cubic Bézier curves”. In: *Computer Aided Geometric Design* 4.1-2 (1987), pp. 23–39 (cit. on pp. 46, 55).
- [SS10] Stefan A Sauter and Christoph Schwab. “Boundary element methods”. In: *Boundary Element Methods*. Springer, 2010, pp. 183–287 (cit. on p. 179).
- [SRN16] Dominik Schillinger, Praneeth K Ruthala and Lam H Nguyen. “Lagrange extraction and projection for NURBS basis functions: A direct link between isogeometric and standard nodal finite element formulations”. In: *International Journal for Numerical Methods in Engineering* 108.6 (2016), pp. 515–534 (cit. on p. 101).
- [Sch96] J. Schweitzer. “Analysis and application of subdivision surfaces”. PhD thesis. University of Washington, 1996 (cit. on p. 116).
- [Sco11] Michael A Scott. “T-splines as a design-through-analysis technology”. PhD Thesis. The University of Texas at Austin, 2011 (cit. on p. 109).
- [Sco+11] Michael A Scott, Michael J Borden, Clemens V Verhoosel, Thomas W Sederberg and Thomas JR Hughes. “Isogeometric finite element data structures based on Bézier extraction of T-splines”. In: *International Journal for Numerical Methods in Engineering* 88.2 (2011), pp. 126–156 (cit. on p. 101).
- [Sco+13] Michael A Scott, Robert N Simpson, John A Evans, Scott Lipton, Stephane PA Bordas, Thomas JR Hughes and Thomas W Sederberg. “Isogeometric boundary element analysis using unstructured T-splines”. In: *Computer Methods in Applied Mechanics and Engineering* 254 (2013), pp. 197–221 (cit. on pp. 84, 132).
- [Sed+03] Thomas W Sederberg, Jianmin Zheng, Almaz Bakenov and Ahmad Nasri. “T-splines and T-NURCCs”. In: *ACM Transactions on Graphics*. Vol. 22. 3. ACM. 2003, pp. 477–484 (cit. on p. 84).
- [Sed+98] Thomas W Sederberg, Jianmin Zheng, David Sewell and Malcolm A Sabin. “Non-uniform recursive subdivision surfaces”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM. 1998, pp. 387–394 (cit. on p. 84).
- [Sei93] Hans-Peter Seidel. “An introduction to polar forms”. In: *Computer Graphics and Applications, IEEE* 13.1 (1993), pp. 38–46 (cit. on p. 17).
- [SK16] Graham Sellers and John Kessenich. *Vulkan programming guide: The official guide to learning vulkan*. Addison-Wesley Professional, 2016 (cit. on p. 140).

- [Sev09] Rubén Sevilla. “NURBS-enhanced finite element method (NEFEM)”. PhD Thesis. Universitat Politècnica de Catalunya, 2009 (cit. on p. 134).
- [SFH08] Rubén Sevilla, Sonia Fernández-Méndez and Antonio Huerta. “NURBS-enhanced finite element method (NEFEM)”. In: *International Journal for Numerical Methods in Engineering* 76.1 (2008), pp. 56–83 (cit. on pp. 88, 133, 134).
- [SFH11] Rubén Sevilla, Sonia Fernández-Méndez and Antonio Huerta. “3D NURBS-enhanced finite element method (NEFEM)”. In: *International Journal for Numerical Methods in Engineering* 88.2 (2011), pp. 103–125 (cit. on p. 134).
- [SS87] Leon A Shirman and Carlo H Séquin. “Local surface interpolation with Bézier patches”. In: *Computer Aided Geometric Design* 4.4 (1987), pp. 279–295 (cit. on p. 46).
- [SS90] Leon A Shirman and Carlo H Séquin. “Local surface interpolation with shape parameters between adjoining Gregory patches”. In: *Computer Aided Geometric Design* 7.5 (1990), pp. 375–388 (cit. on pp. 58, 61).
- [Si15] Hang Si. “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator”. In: *ACM Trans. Math. Softw.* 41.2 (2015), 11:1–11:36 (cit. on pp. 126, 135).
- [SS15] J Smith and Scott Schaefer. “Selective Degree Elevation for Multi-Sided Bézier Patches”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library. 2015, pp. 609–615 (cit. on p. 147).
- [SES04] Jordan Smith, Doug Epps and Carlo Séquin. *Exact evaluation of piecewise smooth Catmull-Clark surfaces using Jordan blocks*. Tech. rep. UC Berkeley, 2004 (cit. on p. 75).
- [Sta98a] Jos Stam. “Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM. 1998, pp. 395–404 (cit. on pp. 74, 78, 116).
- [Sta98b] Jos Stam. “Fast evaluation of Loop triangular subdivision surfaces at arbitrary parameter values”. In: *Computer Graphics (SIGGRAPH’98 Proceedings, CD-ROM Supplement)*. 1998 (cit. on p. 74).
- [Sta01] Jos Stam. “On subdivision schemes generalizing uniform B-spline surfaces of arbitrary degree”. In: *Computer Aided Geometric Design* 18.5 (2001), pp. 383–396 (cit. on p. 82).
- [STZ14] Timothy Sun, Papoj Thamjaroenporn and Changxi Zheng. “Fast multipole representation of diffusion curves and points.” In: *ACM Transactions on Graphics* 33.4 (2014), pp. 53–1 (cit. on p. 179).
- [Sun+12] Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong and Baining Guo. “Diffusion curve textures for resolution independent texture mapping.” In: *ACM Transactions on Graphics* 31.4 (2012), pp. 74–1 (cit. on pp. 179, 180).
- [SPG08] Alok Sutradhar, Glaucio Paulino and Leonard J Gray. *Symmetric Galerkin boundary element method*. Springer Science & Business Media, 2008 (cit. on p. 88).

- [VRS11] Tamás Várady, Alyn Rockwood and Péter Salvi. “Transfinite surface interpolation over irregular n-sided domains”. In: *Computer-Aided Design* 43.11 (2011), pp. 1330–1340 (cit. on p. 149).
- [VSK16] Tamás Várady, Péter Salvi and György Karikó. “A Multi-sided Bézier Patch with a Simple Control Structure”. In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library. 2016, pp. 307–317 (cit. on p. 25).
- [VZ01] Luiz Velho and Denis Zorin. “4–8 Subdivision”. In: *Computer Aided Geometric Design* 18.5 (2001), pp. 397–427 (cit. on pp. 61, 82).
- [VK18] Teun W Verstraaten and Jiří Kosinka. “Local and hierarchical refinement for subdivision gradient meshes”. In: *Computer Graphics Forum*. Vol. 37. 7. Wiley Online Library. 2018, pp. 373–383 (cit. on p. 173).
- [VY14] Antoine Vigneron and Lie Yan. “A faster algorithm for computing motorcycle graphs”. In: *Discrete & Computational Geometry* 52.3 (2014), pp. 492–514 (cit. on p. 117).
- [Vla+01] Alex Vlachos, Jörg Peters, Chas Boyd and Jason L Mitchell. “Curved PN triangles”. In: *Proceedings of the 2001 symposium on Interactive 3D graphics*. Citeseer. 2001, pp. 159–166 (cit. on pp. 46, 147).
- [Vor03] Henk A van der Vorst. *Iterative Krylov methods for large linear systems*. Vol. 13. Cambridge University Press, 2003 (cit. on p. 96).
- [Vuo+11] Anh-Vu Vuong, Carlotta Giannelli, Bert Jüttler and Bernd Simeon. “A hierarchical approach to adaptive local refinement in isogeometric analysis”. In: *Computer Methods in Applied Mechanics and Engineering* 200.49–52 (2011), pp. 3554–3567 (cit. on p. 191).
- [Wac75] Eugene L Wachspress. *A Rational Finite Element Basis*. Academic Press, 1975 (cit. on p. 145).
- [Wal15] Shawn W Walker. *The shape of things: a practical guide to differential geometry and the shape derivative*. Vol. 28. SIAM, 2015 (cit. on p. 88).
- [WM96] Desmond J Walton and Dereck S Meek. “A triangular  $G^1$  patch from boundary curves”. In: *Computer-Aided Design* 28.2 (1996), pp. 113–123 (cit. on pp. 59, 84).
- [Wan09] Lidong Wang. “Integration of CAD and boundary element analysis through subdivision methods”. In: *Computers & Industrial Engineering* 57.3 (2009), pp. 691–698 (cit. on p. 132).
- [Wan+08] Wenping Wang, Yang Liu, Dongming Yan, Bin Chan, Ruotian Ling and Feng Sun. *Hexagonal meshes with planar faces*. Tech. rep. The University of Hong Kong, 2008 (cit. on p. 193).
- [Wan+15] Xiaoning Wang, Xiang Ying, Yong-Jin Liu, Shi-Qing Xin, Wenping Wang, Xianfeng Gu, Wolfgang Mueller-Wittig and Ying He. “Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes”. In: *Computer-Aided Design* 58 (2015), pp. 51–61 (cit. on p. 193).
- [WW01] Joe Warren and Henrik Weimer. *Subdivision methods for geometric design: A constructive approach*. Morgan Kaufmann, 2001 (cit. on pp. 61, 71, 106).

- [Wat88] Michael A Watkins. “Problems in geometric continuity”. In: *Computer-aided design* 20.8 (1988), pp. 499–502 (cit. on pp. 46, 53).
- [WHP11] Anna Wawrzinek, Klaus Hildebrandt and Konrad Polthier. “Koiter’s Thin Shells on Catmull-Clark Limit Surfaces.” In: *VMV*. 2011, pp. 113–120 (cit. on p. 103).
- [WP16] Anna Wawrzinek and Konrad Polthier. “Integration of generalized B-spline functions on Catmull-Clark surfaces at singularities”. In: *Computer-Aided Design* (2016) (cit. on pp. 106, 116).
- [Wei+16] Xiaodong Wei, Yongjie Jessica Zhang, Thomas JR Hughes and Michael A Scott. “Extended truncated hierarchical Catmull-Clark subdivision”. In: *Computer Methods in Applied Mechanics and Engineering* 299 (2016), pp. 316–336 (cit. on p. 125).
- [Wei+15] Xiaodong Wei, Yongjie Zhang, Thomas JR Hughes and Michael A Scott. “Truncated hierarchical Catmull-Clark subdivision with local refinement”. In: *Computer Methods in Applied Mechanics and Engineering* 291 (2015), pp. 1–20 (cit. on pp. 125, 191).
- [Wei88] Kevin Weiler. “The radial edge structure: a topological representation for non-manifold geometric boundary modeling”. In: *Geometric modeling for CAD applications* (1988), pp. 3–36 (cit. on p. 194).
- [Wei16] Martin Weiser. *Inside finite elements*. Walter de Gruyter GmbH & Co KG, 2016 (cit. on p. 88).
- [Wij86] Jarke J van Wijk. “Bicubic patches for approximating non-rectangular control-point meshes”. In: *Computer Aided Geometric Design* 3.1 (1986), pp. 1–13 (cit. on pp. 46, 51).
- [WA02] George Wolberg and Itzik Alfy. “An energy-minimization framework for monotonic cubic spline interpolation”. In: *Journal of Computational and Applied Mathematics* 143.2 (2002), pp. 145–188 (cit. on p. 160).
- [Wol18] David Wolff. *OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++ 17*. Packt Publishing Ltd, 2018 (cit. on p. 140).
- [Yan+17] Zhipei Yan, Stephen Schiller, Gregg Wilensky, Nathan Carr and Scott Schaefer. “ $\kappa$ -curves: interpolation at local maximum curvature”. In: *ACM Transactions on Graphics* 36.4 (2017), p. 129 (cit. on p. 158).
- [ZSC18] Qiaoling Zhang, Malcolm A Sabin and Fehmi Cirak. “Subdivision surfaces with isogeometric analysis adapted refinement weights”. In: *Computer-Aided Design* 102 (2018), pp. 104–114 (cit. on pp. 78, 116).
- [ZDZ17] Shuang Zhao, Fredo Durand and Changxi Zheng. “Inverse Diffusion Curves using Shape Optimization”. In: *IEEE Transactions on Visualization and Computer Graphics* (2017) (cit. on p. 180).
- [ZL05] Xianlian Zhou and Jia Lu. “NURBS-based Galerkin method and application to skeletal muscle modeling”. In: *Proceedings of the 2005 ACM symposium on Solid and physical modeling*. ACM. 2005, pp. 71–78 (cit. on p. 126).

- [ZTZ05] Olgierd C Zienkiewicz, Robert L Taylor and Jian Z Zhu. *The finite element method: its basis and fundamentals*. Elsevier, 2005 (cit. on pp. 88, 92, 96, 97, 126).
- [ZJK16] Urška Zore, Bert Jüttler and Jiří Kosinka. “On the linear independence of truncated hierarchical generating systems”. In: *Journal of Computational and Applied Mathematics* 306 (2016), pp. 200–216 (cit. on pp. 125, 191).
- [Zwa73] Philip B Zwart. “Multivariate splines with nondegenerate partitions”. In: *SIAM Journal on Numerical Analysis* 10.4 (1973), pp. 665–673 (cit. on p. 29).



## Acknowledgement

Thanks to the anonymous reviewers for — wait, this is not a journal paper! In that case, where do I start? And in which order... Perhaps chronologically, based on social/personal distance, or maybe by Erdős number? A blend of all three might be best. In that case, I would like to start with the people academically closest to me — my supervisors Jiří and Jos.

Jiří, it has been a great (and at times crazy) four years, even longer if we also consider our adventures back in the UK. A heartfelt thank you for all your insight, feedback, patience, and much more! For simultaneously (subtly) discouraging *and* proposing side topics and activities. For your company during the conferences we attended together (some sessions which might have been spent in a hammock or tropical pool). For showing that there are still higher levels of perfection. And for the good times outside of work, including tea sessions, LEGO events and sports (we still have to go for a squash rematch). You are an excellent researcher with a clear passion for teaching and supervision (I recommend getting a larger — or second — whiteboard though), and I wish you the best of luck for your future career! You have become a dear friend, and I am convinced our paths will continue to intersect. *Děkuji!*

Jos, although our interaction was mostly limited to the occasional progress meeting, thank you for sharing your thoughts and experience these past four years.

Naturally, there are several other people in a scientific context to whom I would like to extend my gratitude. Malcolm, the number of different topics you have worked on — often at least a decade before others considered it — never ceases to amaze me. Thank you for your guidance back in Cambridge, and for our correspondence by email once in a while. It is much appreciated. Michael, *eskerrik asko* for having me at BCAM (Bartoň's Computer Aided Manufacturing) for a couple of weeks! I am looking forward to future collaborations. Jörg, thanks for exchanging thoughts on  $G^1$ - and subdivision-related matters. And thanks to many others I had the pleasure of meeting (and discussing a wide variety of topics with) during a workshop or conference.

Then on to the PhD assessment- and examining committees. Thank you Kai, Gert and Alex for constituting the former, and Fred and Kerstin for complementing the latter. I am looking forward to my defence and the events following it!

Another important part of a (Dutch) PhD defence are the paranymphs. Georg and Youngjoo, thanks a lot for taking on these ceremonial tasks! And of course also for the great board game- and Nintendo Switch nights, frisbee sessions and other entertainment :).

Cheers also to the (former) postdocs and fellow PhD candidates at SVCG — Jasper for helping me out with questions mostly related to programming, Matthew for the LEGO/BrickLink distractions, Renato and Paulo for the games of pool, Venus for the occasional Spanish/Dutch language exchange, Chengtao for the tasty Chinese dinners, André and Han for the good conversations, Gerben for the spline-related discussions and everyone else for the incidental chat during lunch or some other time. Also a word of thanks to the secretaries and other staff, in particular Desiree († 2018), Ineke, Heleen, Janieta and Martin.

The PhD community in Groningen obviously extends far beyond one's own research group. Pry, thanks for being my sporting buddy (and for pushing me to enrol in a squash course!), Artem for sharing my love for tea, and cheers to many other people from the Allersmaborg group and the mastering-your-PhD course (also known as group therapy). Frank, Loek, Ahmad, Morten, Bin, Elwin, Nikky and Richard — I had a great time jamming, rehearsing songs and eventually performing some of them! Gabriel, *obrigado* for introducing me to some great new bands.

Although occasionally hidden from us, there also appears to be life outside of academia. Thanks to the enthusiastic people from FabLab Groningen for making it possible to work with all sorts of cool equipment! People from the Japanese Culture and Language group, どうもありがとう for the nice events and conversations, especially Mami-san and Takuya-san.

Pap en mam, waar te beginnen? Pap, met je gezonde afkeer van alles dat met wiskunde te maken heeft. Ik herken steeds meer van jou in mezelf, en dan bedoel ik niet alleen de gedeelde interesses voor bijvoorbeeld de tuin. Helaas niet meer helder genoeg om deze promotie bewust mee te maken, maar in gedachte ben je er zeker bij. Mam, zoveel waardering voor een onbezorgde start, aanmoediging tot creativiteit en nog veel meer. Wat een geduld en geheugen, daar kan ik alleen maar bewondering voor hebben! Roos, voor het valsspelen met Monopoly — geintje natuurlijk :D. Best fijn om zo'n grote zus te hebben, zeker als je er muzikaal door wordt geïnspireerd en opeens besluit om te gaan drummen! Proost op een goede toekomst in Wageningen, een gezonde dosis bordspellen en geneuzel over plantjes — uiteraard samen met Rutger! Pienie, misschien dan wel de jongste, maar een portie discipline en onafhankelijkheid waar je u tegen zegt. Supertrots op wat je allemaal doet! En die moderne kunst, misschien leer ik het ooit nog wel te waarderen ;).

Als ik vervolgens onderzoek en familie combineer, kom ik uit bij mijn oma Jeanne Detollenaere († 2004). De interesse voor vormen en patronen komt ongetwijfeld van die kant (de vele quilts en ander handwerk), en daarnaast was het bij mijn grootouders dat ik mijn eerste computervaardigheden opdeed! Wat zou het leuk zijn geweest om mijn onderzoek met u te bespreken...

Over vormen en patronen gesproken, bedankt mevrouw Baalman voor de fijne wiskundelessen op de middelbare school (vooral die over meetkunde) — absoluut een inspiratie voor de stappen die ik daarna heb gemaakt.

Dan komen we uit in Eindhoven, waar zoveel leuke huisgenoten waren dat ik uiteindelijk zeven jaar in hetzelfde studentenhuus heb gewoond en er veel goede vriendschappen aan heb overgehouden. Dan denk ik vooral aan Frank & Mari — *tānan vāga* voor de goede tijd samen, kampvuurtjes en fijne gesprekken. Tot snel weer in Tallinn, Muhu of elders! Adriaan & Ana als mede-theeleuten en bordspelfanaten — succes in Lyon, A+! Rody (de laatste periode min of meer onderdeel van het meubilair aan de Gestelsestraat) als mede-werktuigbouwer & Matilde, kattenliefhebbers en inmiddels allebei al gepromoveerd. Otto, held in het zelf maken van dingen in een fantastische klusschuur & Eva.

Wat de TU/e betreft, een woord van dank voor Clemens voor de aangename begeleiding en samenwerking, en voor Göktürk voor de discussies over vanalles en nog wat.

Rocío, han pasado tres años desde que nos conocimos, y ya hemos compartido muchos viajes, experiencias y recuerdos. Estoy muy orgulloso de ti — por abrazar la cultura y el idioma holandés (estoy disfrutando mucho de las contrapartes mexicanas), tu talento para dibujar y pintar (¡preparemos *Ayy antojitos!*), probar prácticamente todos los deportes que existen y mucho más. Siempre estás ahí para escucharme y ayudarme a relajar, y admiro tu tolerancia con respecto a los montones de papeles/notas en toda la casa, los cambios de ritmo y otras cosas que pueden no ser siempre agradables. Pronto será tu turno de escribir una disertación, intentaré ser tan buen apoyo como lo has sido. Te quiero mucho, mijn kleine Mango, ¡y espero con ansias nuestras futuras aventuras juntos!

Of course there are also the people I met because of you — Edisa, a best friend of Rocío and a good friend of mine. You have considerably increased my tolerance regarding people dropping by without appointment ;). Xin, 謝謝 for the games of table tennis and the delicious fish! Bimal & Pragy, for your hospitality both in Nepal and here in Groningen, and of course the hand-made momos. Gaby & Oli, for your warmth and friendliness (and for teaching me some Mexican slang, most of which I am not allowed to use at home), and many others.

Finally, thanks to the LINUX, INKSCAPE and BLENDER communities, the multitude of online platforms offering support regarding programming or scripting, and the many people pursuing and enabling unlimited access to knowledge worldwide.